EWCK-STM32 User Guide

- STM32 개발환경 구축 및 I/O 제어 실습 -

EWCK-STM32 개발 보드를 구입해 주신 고객님께 진심으로 감사드립니다. 본 User Guide는 EWCK-STM32를 구입하신 분들께서 EWCK-STM32 보드를 이용하여, 편리하고 쉽게 제품을 개발하실 수 있도록 도와드리기 위해서 제작되었습니다.

본 문서의 모든 저작권은 (주)지비테크에 있습니다. 무단 전제 및 복제를 금합니다.
Copyright© 2010 by GBtec Inc. LTD.

▣ Contents

▣ 하드웨어 구성 품목 ····· 4
▣ 보드 커넥터 명칭 ····· 7

실습 1. 개발환경 구축하기 ····· 8
1.1 컴파일러 설치하기 ····· 8
1.2 Eclipse 설치하기 ····· 10
1.3 Eclipse 실행 및 프로젝트 설정하기 ····· 12
1.4 FlashLoader 설치하기 ····· 25

실습 2. GPIO 제어하기 ····· 33
2.1 GPIO (General Purpose Input/Output) 개요 ····· 33
2.2 GPIO 레지스터 ····· 34
2.3 GPIO 라이브러리 함수 ····· 37
2.4 LED 제어 실습하기 ····· 40
2.5 Key 제어 실습하기 ····· 42

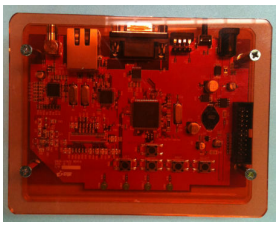
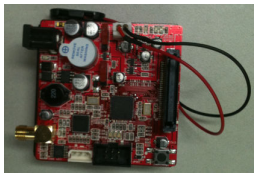
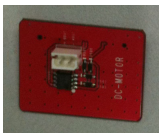
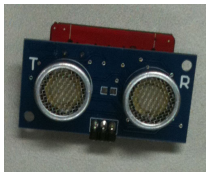
실습 3. GPIO Interrupt 제어하기 ····· 45
3.1 GPIO Interrupt 개요 ····· 45
3.2 GPIO Interrupt 레지스터 ····· 47
3.3 GPIO Interrupt 라이브러리 함수 ····· 49
3.4 GPIO Interrupt 제어 실습하기 ····· 51

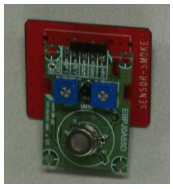
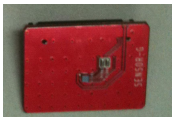
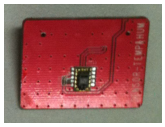
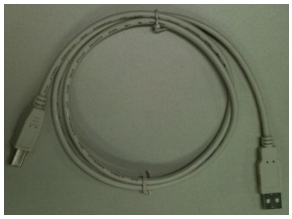
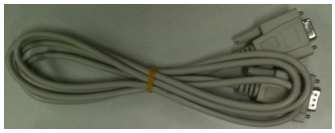
실습 4. USART 통신 제어하기 ····· 55
4.1 USART 통신 개요 ····· 55
4.2 USART 레지스터 ····· 56
4.3 USART 라이브러리 함수 ····· 59
4.4 USART 통신 제어 실습하기 ····· 61

실습 5. Timer 제어하기 ····· 66
5.1 Timer 개요 ····· 66
5.2 Timer 레지스터 ····· 68
5.3 Timer 라이브러리 함수 ····· 73
5.4 Timer 제어 실습하기 ····· 76

실습 6. SPI 제어하기 ····· 83
6.1 SPI(Serial Peripheral Interface) 개요 ····· 83
6.2 SPI 레지스터 ····· 86
6.3 SPI 라이브러리 함수 ····· 90

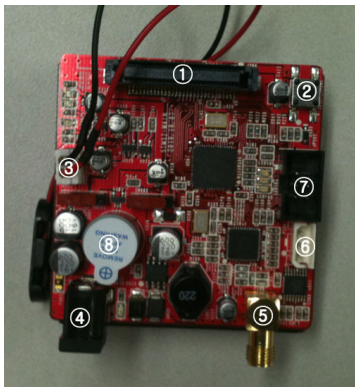
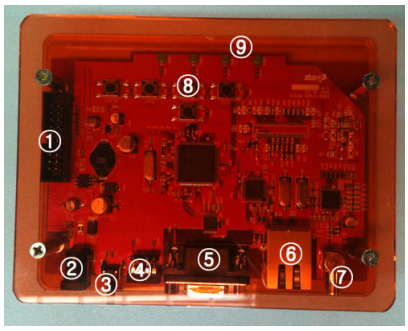
▣ 하드웨어 구성 품목

번호	품목		수량	비고
1		서버 메인 보드	1	MCU : STM32f107vct
2		센서 메인 보드	1	MCU : ATmega128L
3		PIR 센서	1	
4		DC 모터	1	
5		Setp 모터	1	
6		GPS	1	
7		초음파 센서	1	
8		조도 센서	1	

9		연기 센서	1	
10		G 센서	1	
11		온습도 센서	1	
12		USB ISP	1	
13		ISP 커넥터	1	
14		ISP USB 케이블	1	
15		12V 2.5A 어댑터	2	
16		RS232C 통신 케이블	1	
17		3 Pin RS232C 통신 케이블	1	

18		이더넷 케이블	1	
19		안테나	3	
20		Stepping 모터	1	
21		DC 모터	1	
22		모터용 바퀴	2	
23		충전기	1	
24		충전용 배터리	4	
25		데이터 CD	1	1. 예제소스 2. 개발 툴 3. 실습 교재 4. 회로도 5. 데이터 시트

▣ 보드 커넥터 명칭

보드	연결부	설명
센서 메인 보드		① 센서 보드 커넥터 ② Reset 버튼 ③ 배터리 커넥터 ④ 전원 커넥터 ⑤ 안테나 커넥터 ⑥ 3Pin RS232C 커넥터 ⑦ ISP 커넥터 ⑧ Buzzer
서버 메인 보드		① JTag 커넥터 ② 전원 커넥터 ③ Reset 버튼 ④ Booting Mode 선택 스위치 ⑤ RS232C 커넥터 ⑥ 이더넷 커넥터 ⑦ 안테나 커넥터 ⑧ Debuging 스위치 (5EA) ⑨ Debuging LED (4EA)

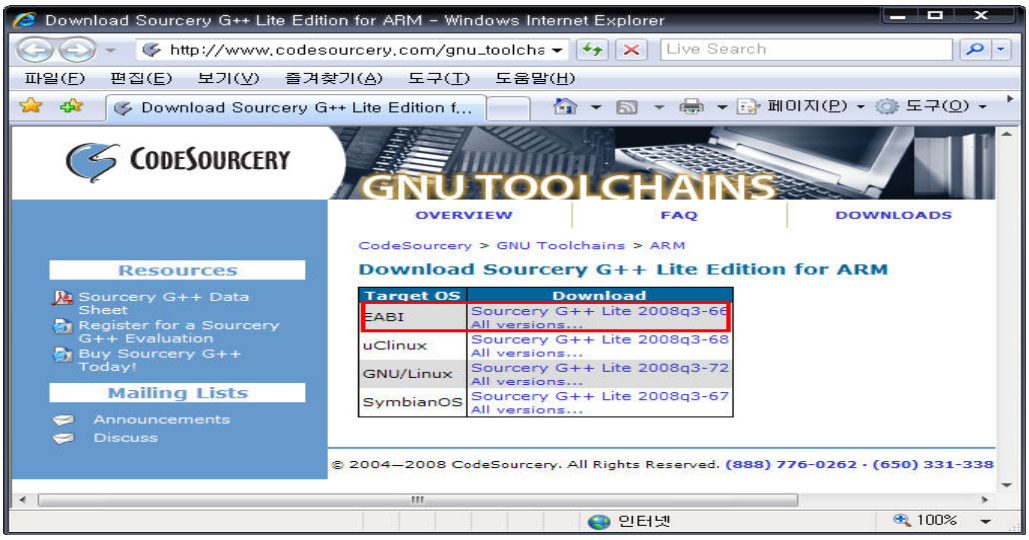
실습 1. 개발환경 구축하기

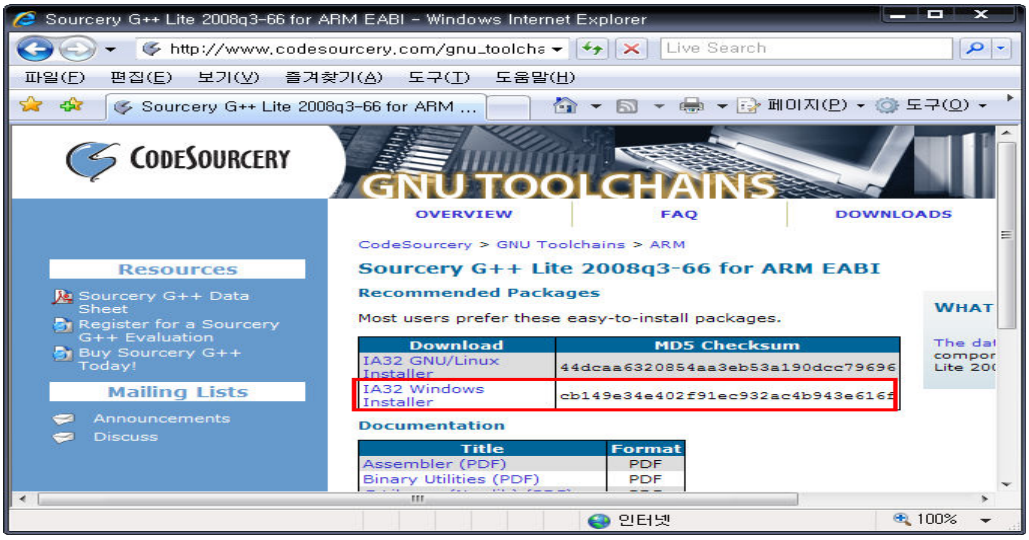
1.1 컴파일러 설치하기

고급 언어로 작성되어진 프로그램 코드는 기계가 바로 실행할 수 없으므로, 기계가 알아들을 수 있는 언어로 변경하는 과정을 거쳐야만 비로서 작성되어진 코드 내용대로 기계가 동작하게 된다. 이러한 과정은 컴파일러라는 별도의 프로그램을 통해 이루어진다. 따라서 실습에 사용되어질 컴파일러인 GCC를 설치하도록 한다.

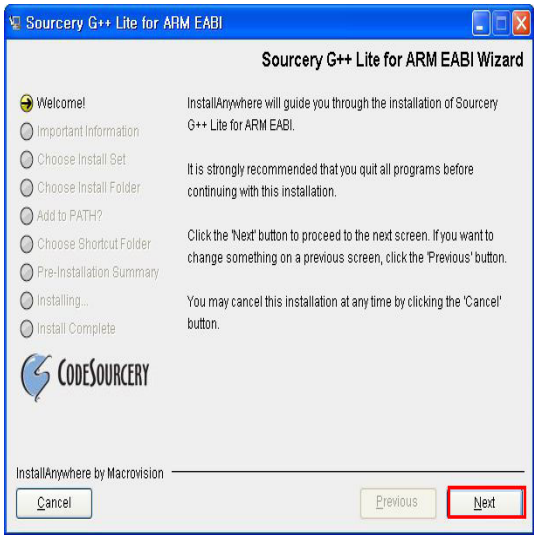
GCC는 여러 플랫폼에 맞는 다양한 버전이 존재지만, 본 실습에서는 Code Sourcery에서 배포하는 CodeSourcery G++ Lite를 다운로드하여 사용하도록 한다. 아래 웹 사이트를 통해 최신 버전의 CodeSourcery G++ 컴파일러를 다운로드 받아 설치하도록 한다.

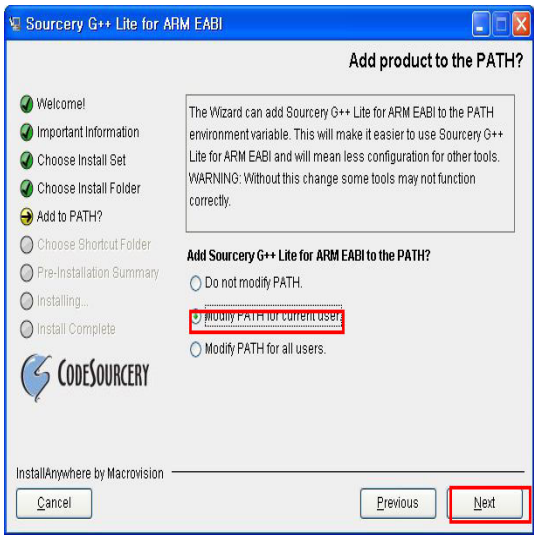
http://www.codesourcery.com/gnu_toolchains/arm/download.html

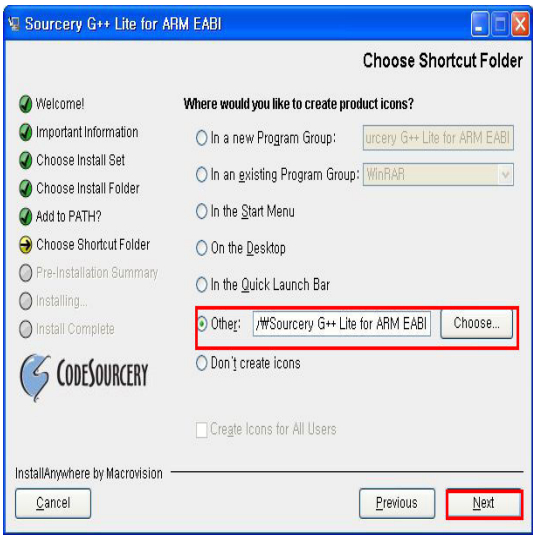




위 그림과 같이 사각형으로 표시한 부분의 CodeCourcery G++ Lite 설치 파일을 다운로드 받고, 설치를 진행하도록 한다. 설치 단계는 아래와 같다.(좌측에서 우측 진행)







위 과정을 마치면 자동으로 해당 폴더에 CodeSourcery G++ Lite가 설치되고, 시스템 환경 변수에 설치경로가 추가되어지며, 이 경로를 이용하여 컴파일러와 링커에 접근하여 사용하게 된다.

1.2 Eclipse 설치하기

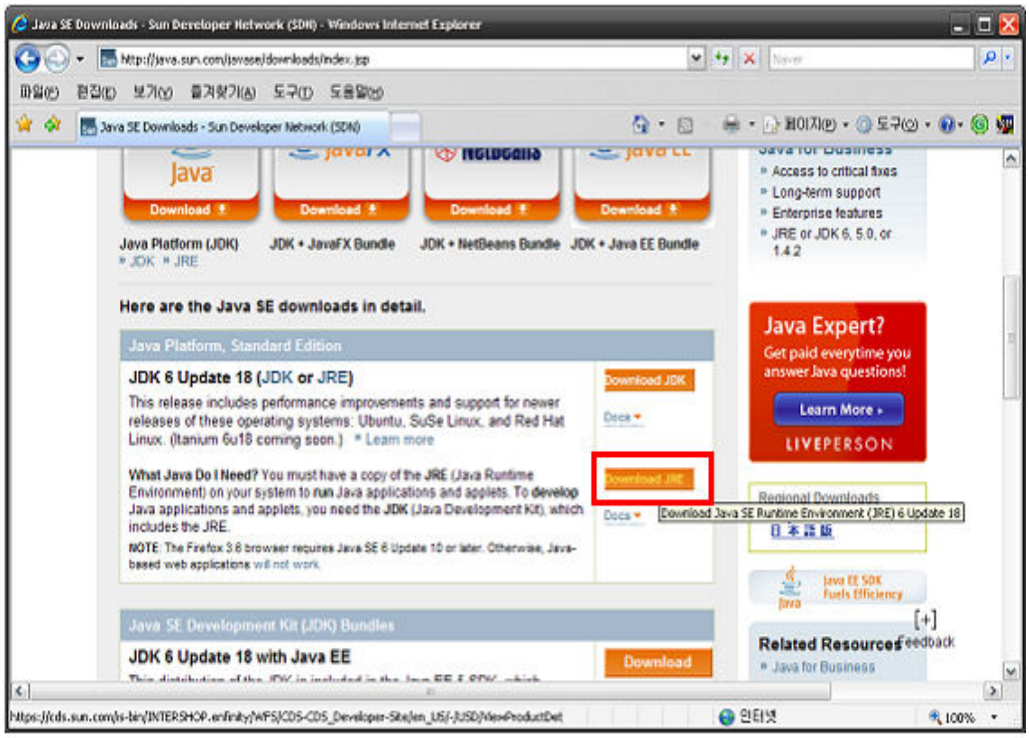
앞으로 실습에 사용할 통합 개발 툴(이하 IDE)은 Eclipse이다. Eclipse는 원래 IBM에서 막대한 예산을 투자해 상용 IDE를 목적으로 개발한 툴이다. 물론 나중에 웹 스피어 스튜디오는 이를 기반으로 만들기는 했지만 IBM은 오픈 소스 단체에 기여한다는 의미에서 이 개발 툴을 오픈 소스로 개방하고 이름을 Eclipse라고 하였다.

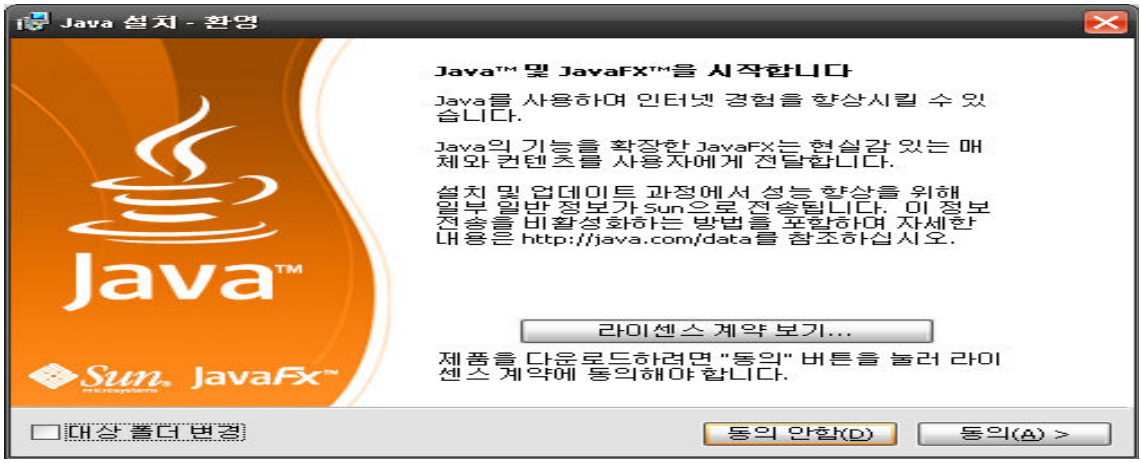
IBM은 Eclipse가 단순한 개발 툴뿐만이 아니고 다른 IDE 개발의 플랫폼이 되도록 설계하였다. 즉 Eclipse를 기반으로 기능을 추가해 다른 IDE를 만들 수 있도록 한 것이다. 이는 Sun사에서 지원하는 NetBeans와 유사하다. NetBeans나 Eclipse나 모두 확장성이 뛰어나고 추가 가능한 모듈이나 플러그인들이 많아 다양한 기능들을 접할 수 있다. 또한 운영체제에 독립적이라는 큰 장점을 지니고 있는 통합 개발 툴로서 자리 매김하고 있다. 이는 Java 기반 IDE로서 JRE가 설치되어 있는 어느 OS 환경에서도 실행이 가능하기 때문이다. Eclipse는 다양한 언어의 프로그래밍을 지원하고 있으며 그 중 본 실습에서 사용할 Eclipse의 버전은 C/C++ 버전을 사용하도록 한다.

Eclipse의 설치는 여타 윈도우 프로그램과 다르게 install이라는 과정이 필요치 않으며, 단지 적당한 폴더에 압축만 풀어주기만 하면 실행 가능한 형태로 이루어져 쉽게 Eclipse를 설치 및 사용이 가능 하다.

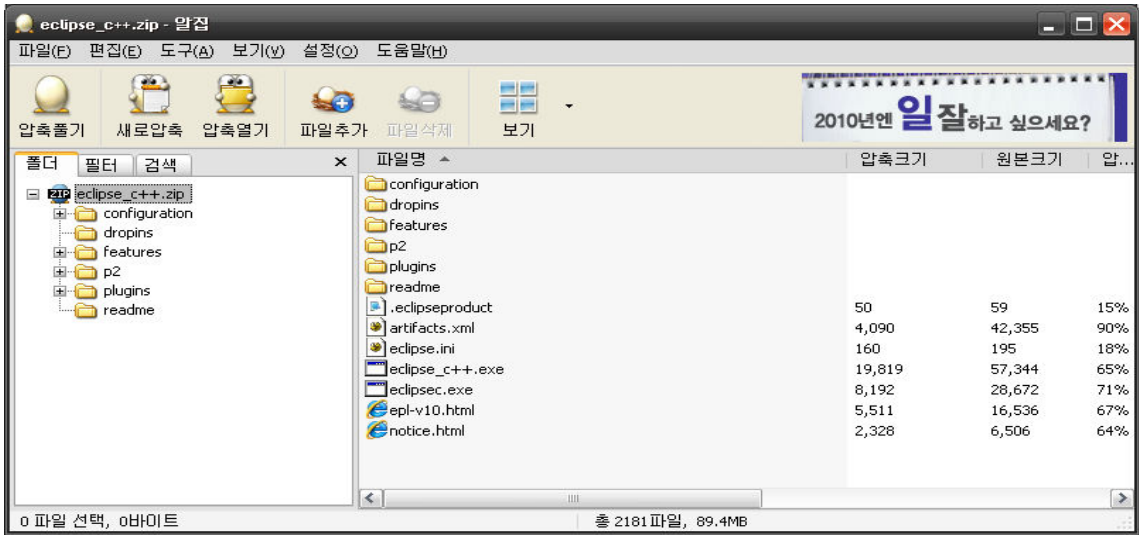
본 실습에서는 CD에 제공되는 Eclipse_C/C++를 사용하여 설치하도록 하겠다. 새로운 버전을 다운로드 받고자하는 경우 http://eclipse.org 에서 다운로드 받아 사용하여도 무방하다.

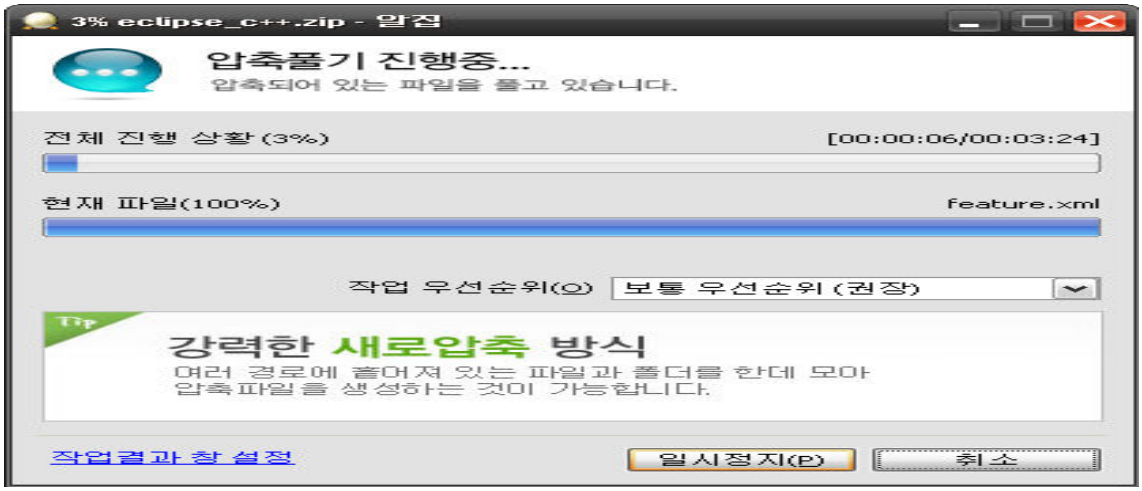
Eclipse 설치에 앞서 위에서 언급했듯이 Eclipse는 Java 기반에서 동작하는 IDE이므로 JRE라는 실행과 관련된 라이브러리를 해당 PC에 설치를 해주어야만 실행이 가능하다. JRE는 http://java.sun.com.javase/downloads/index.jsp 에서 다운로드 받아 설치하도록 한다.



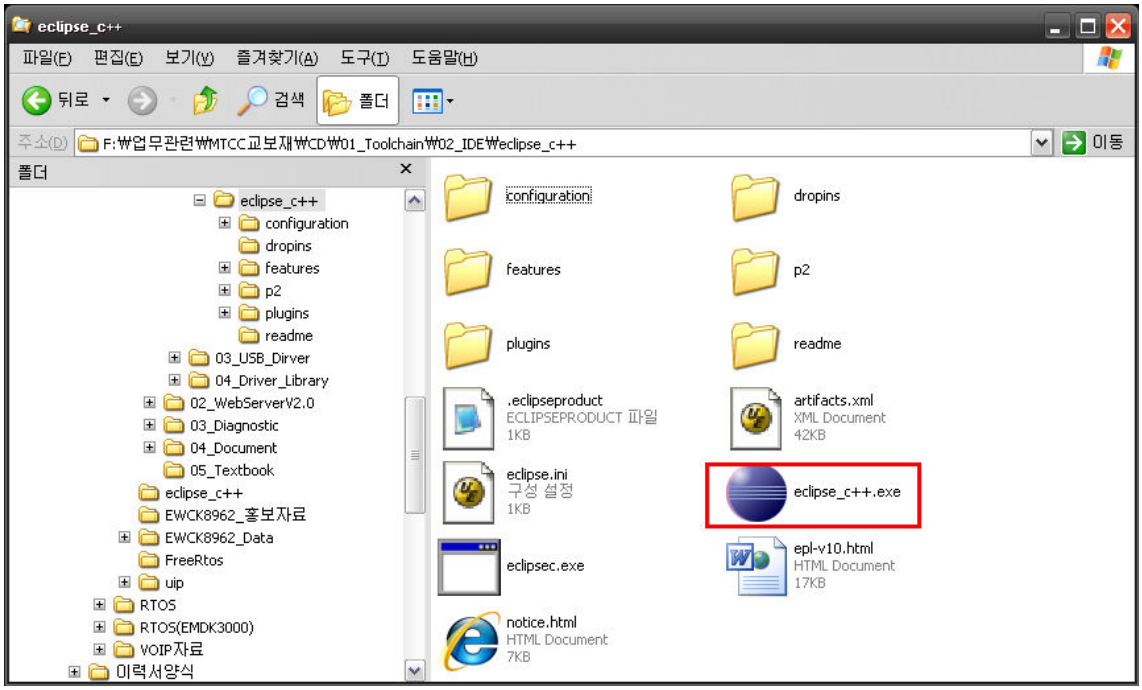


설치 방법은 다운로드 받은 설치파일을 실행해 [Next] 버튼만 설치가 완료 될 때까지 눌러주면 된다. 해당 PC에 JRE가 설치되어 있는 경우 별도의 설치는 이루어지지 않으며, 설치 단계는 바로 끝나게 된다. JRE의 설치가 완료되었다면 이제 Eclipse를 CD에서 복사해서 적당한 위치에 압축을 풀도록 한다.



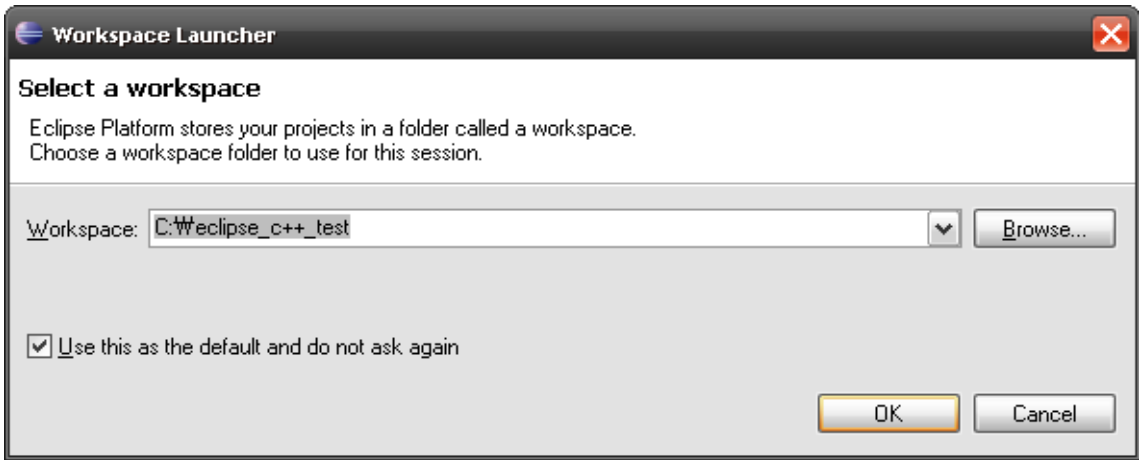


Eclipse 압축 풀기를 완료하고 난 후, 사용을 편리하게 하기위해 바탕화면에 단축 아이콘을 생성한다. 먼저 압축을 푼 폴더로 이동해 실행 파일인 eclipse_c++.exe 파일을 선택해 바탕화면이나 단축 창으로 드래그&드랍하여 바로가기 아이콘을 생성하도록 한다.



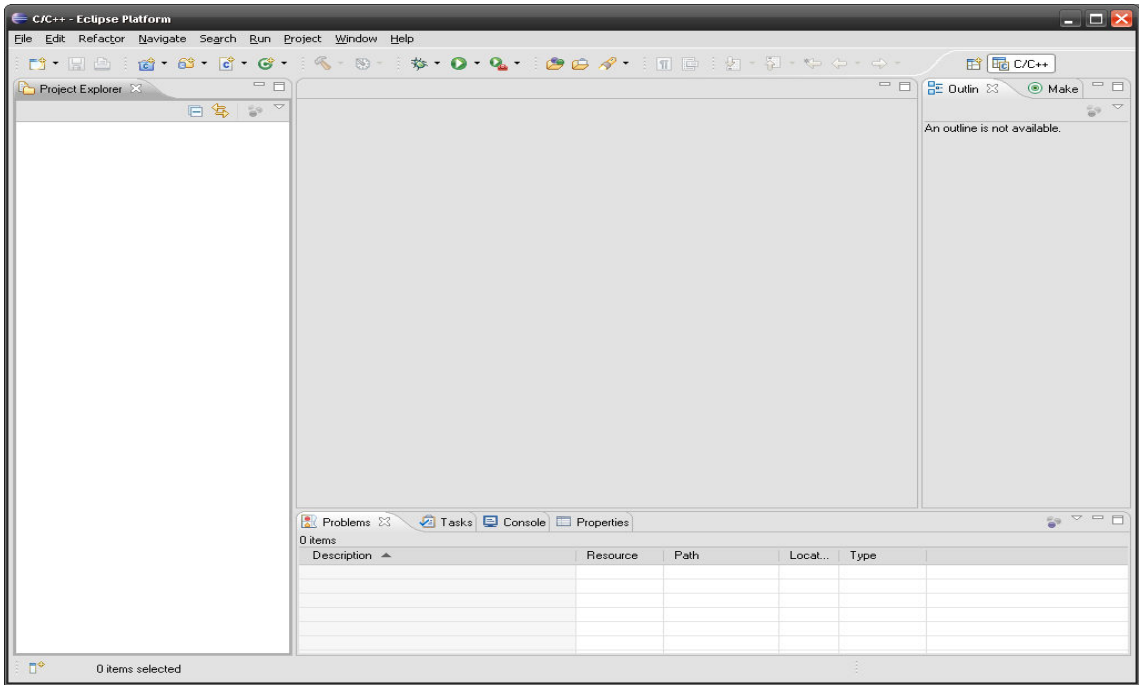
1.3 Eclipse 실행 및 프로젝트 설정하기

Eclipse를 이용해 프로젝트를 생성하기 위해 먼저 Eclipse를 실행하도록 한다. 실행 시 가장 먼저 작업 공간 설정 창이 화면에 나타나게 되는데, 이는 앞으로 만들어질 프로젝트들을 저장할 최상위 폴더를 의미하게 된다. 자신이 찾기 쉬운 장소를 정해 작업 공간의 위치를 정하고 [OK] 버튼을 눌러 다음으로 진행한다.

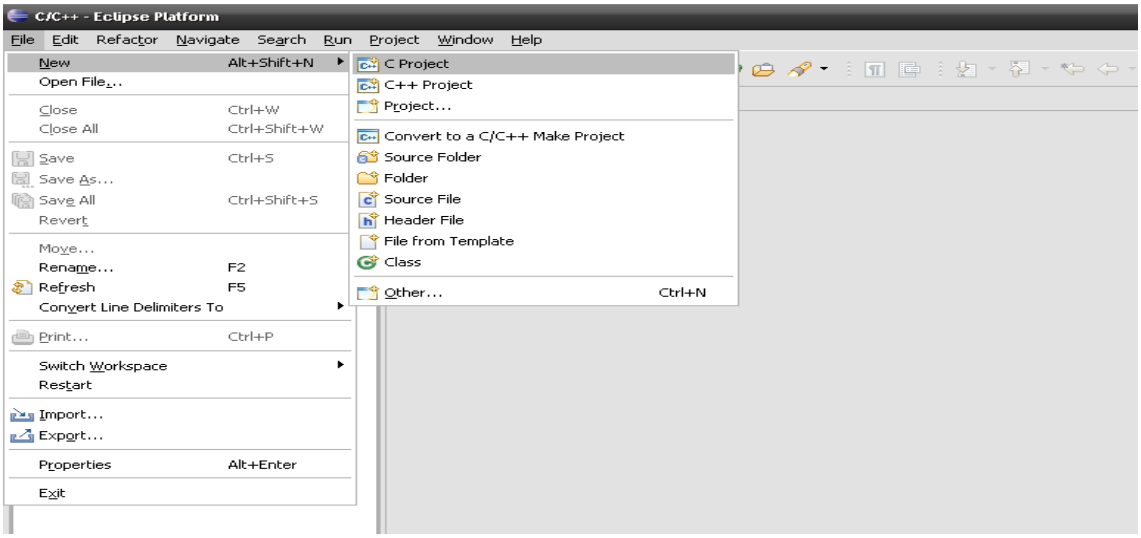


작업 공간 선택 후 나타나는 화면으로 Eclipse의 메인 화면이며, 최초 실행 시 나타나는 화면이다. 화면에 보여 지는 별이 들어있는 원, 지구본이 들어있는 원 등은 Eclipse를 이용하는데 있어 정보를 제공하는 메뉴이므로 한 번씩 눌러서 확인해보도록 한다.

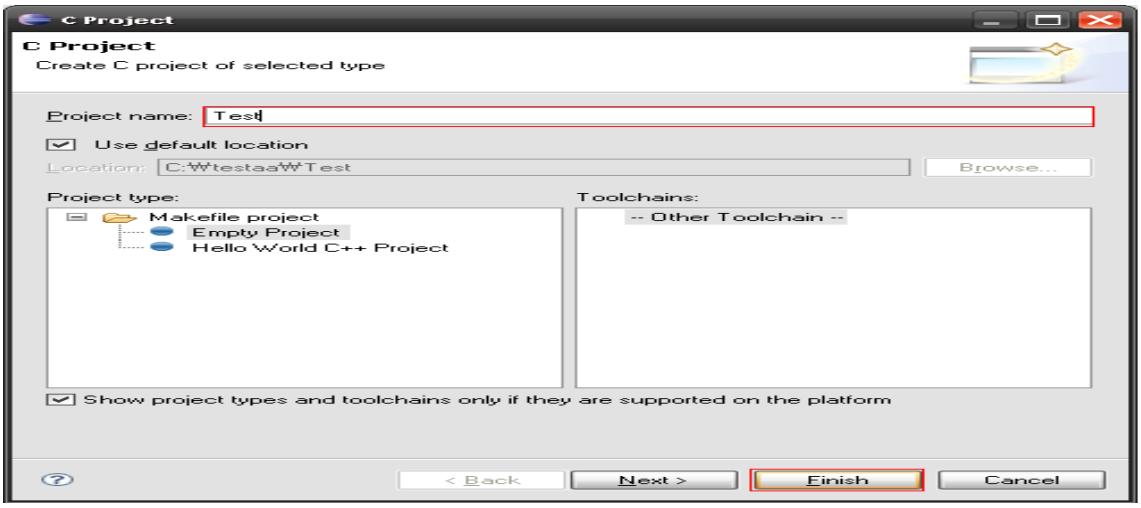
위 화면에서 Welcome 탭의 'X' 표시를 눌러 탭을 닫게 되면 아래와 같이 화면이 바뀌게 되는데, 이 화면 구성이 앞으로 실질적인 작업을 하게 될 화면의 구성이 된다.

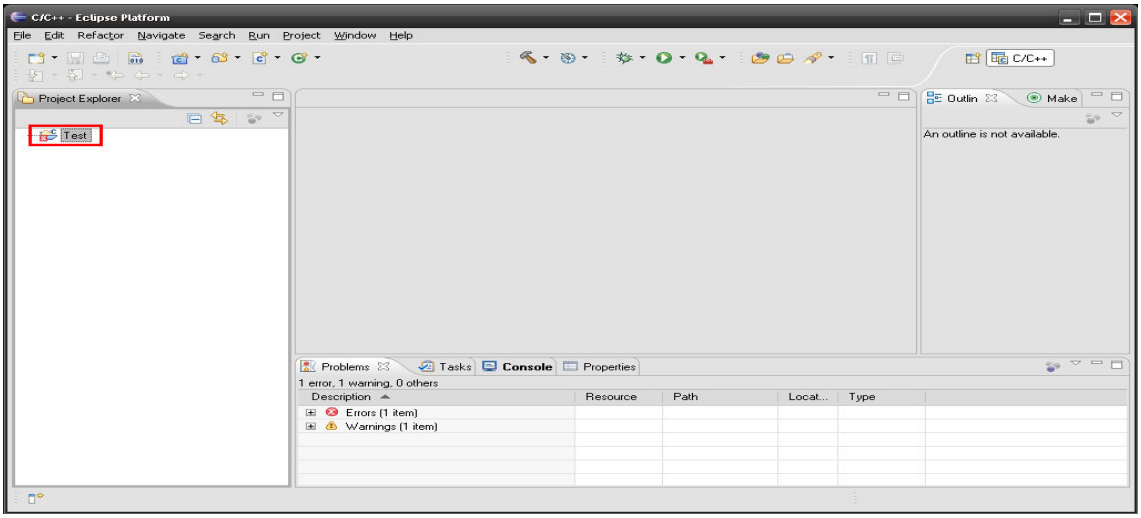


지금부터 Eclipse를 이용한 프로젝트 생성 방법에 대해서 알아보도록 한다. 프로젝트 생성 순서는 아래 그림을 참조하여 차근차근 진행하도록 한다. 우선 Eclipse의 메뉴에서 File->New->C Project를 선택한다.

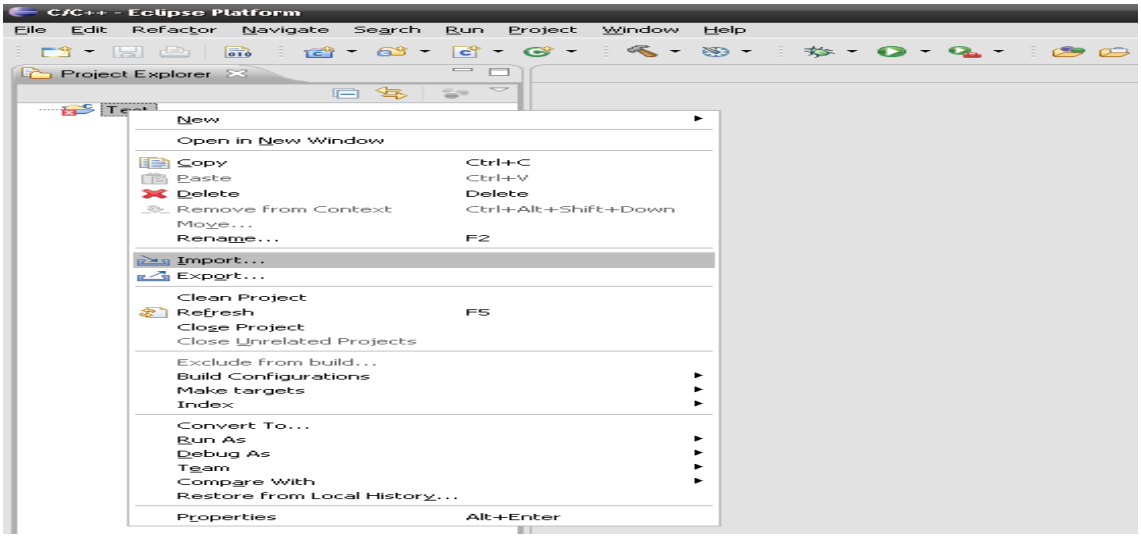


Project Name에 생성하고자하는 프로젝트 이름을 작성하고 [Finish] 버튼을 누르게 되면 아래와 같이 Test라는 이름의 프로젝트가 생성되어진 것을 확인할 수 있다.

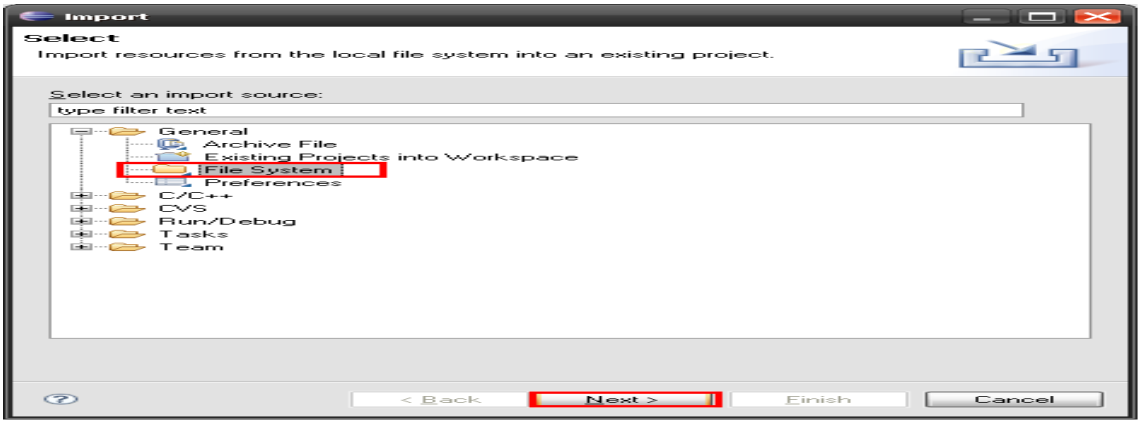




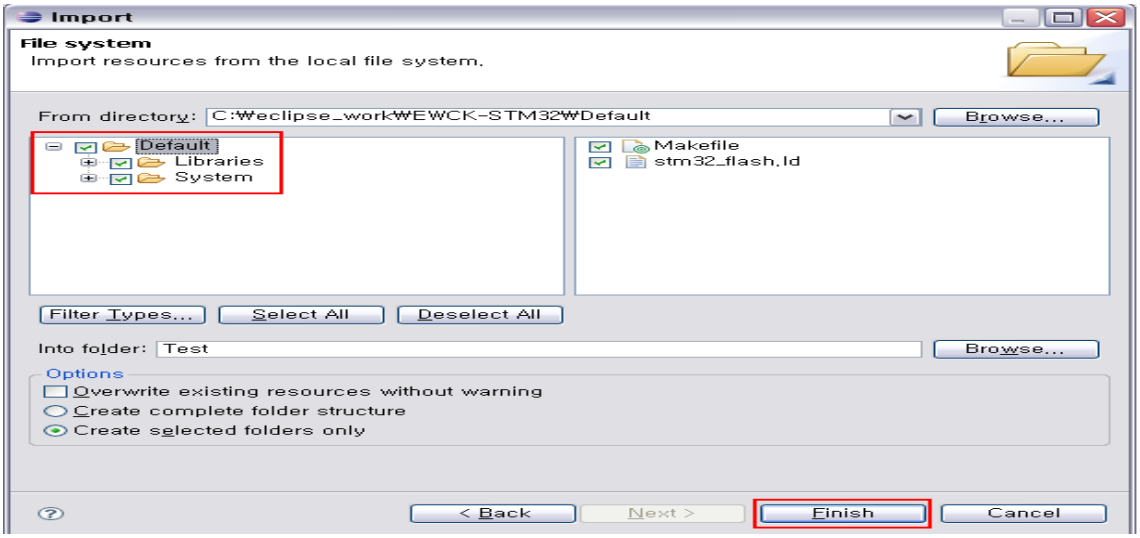
다음으로 제어 프로그램을 개발하기위한 필수 파일들을 프로젝트에 추가한다. 위에서 생성한 프로젝트 이름 위에서 마우스 오른쪽 버튼을 눌러 나타나는 팝업 메뉴 중 [Import]를 선택한다.

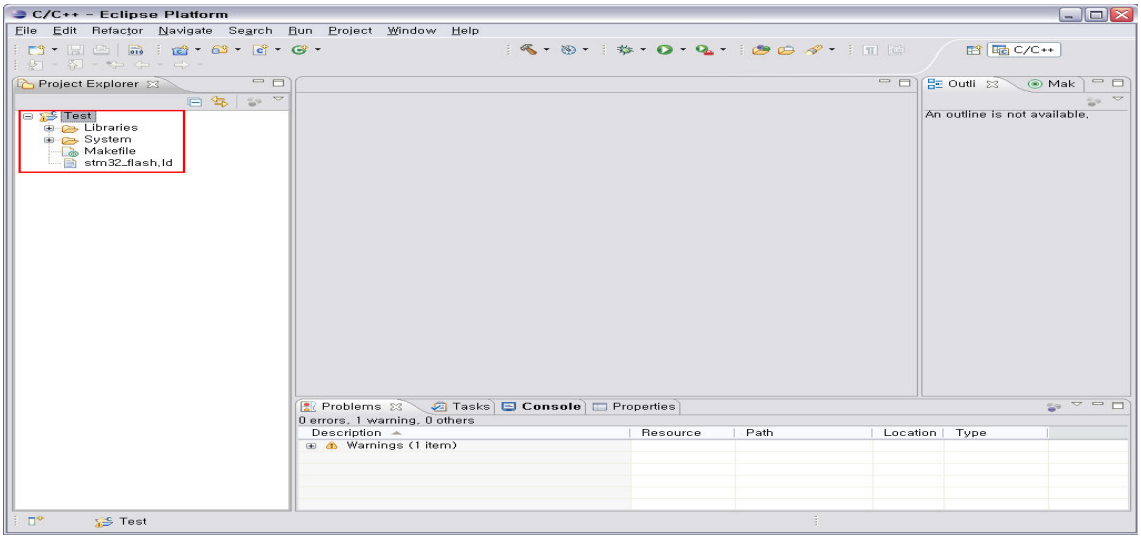


General에서 File System을 선택하고, [Next] 버튼을 눌러 다음으로 이동한다.



Browse... 버튼을 눌러 CD안의 Source -> Default 폴더를 선택하고, 아래 그림과 같이 기본적으로 포함되어야 할 파일들을 선택한 후 [Finish] 버튼을 누르게 되면 프로젝트에 추가된 파일들을 볼 수 있다.





지금까지의 과정은 EWCK-STM32 기본 Firmware 라이브러리, 컴파일 스크립트 및 링커 스크립트 파일을 추가하는 과정으로 주변 I/O 제어 및 컴파일을 하기위해서는 필수 파일들을 추가하는 과정이므로, 새로운 프로젝트를 생성할 경우 반드시 선행되어야한다.

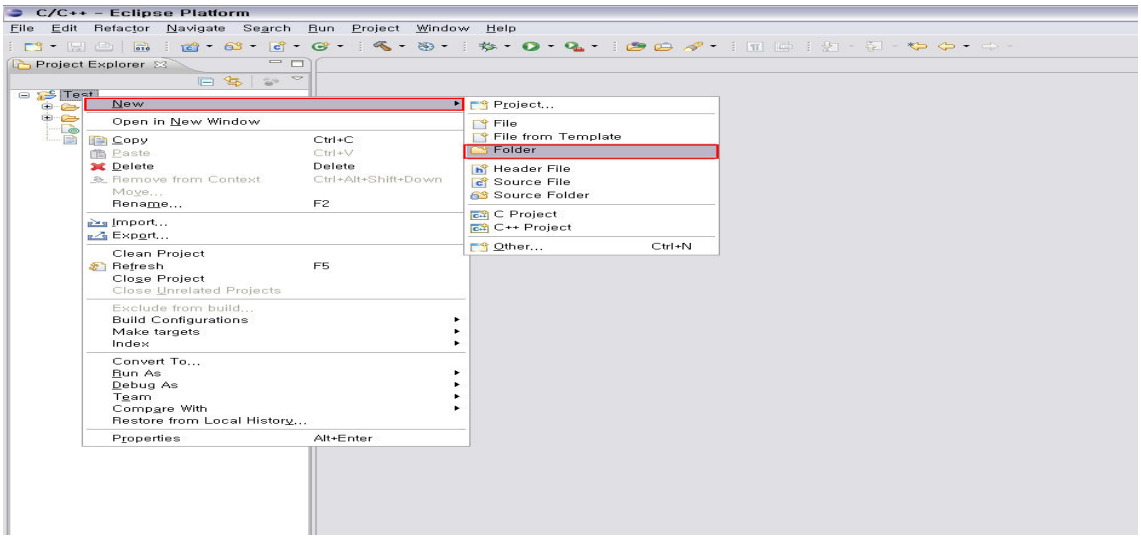
추가되어진 파일의 기능은 아래와 같이 요약할 수 있다.

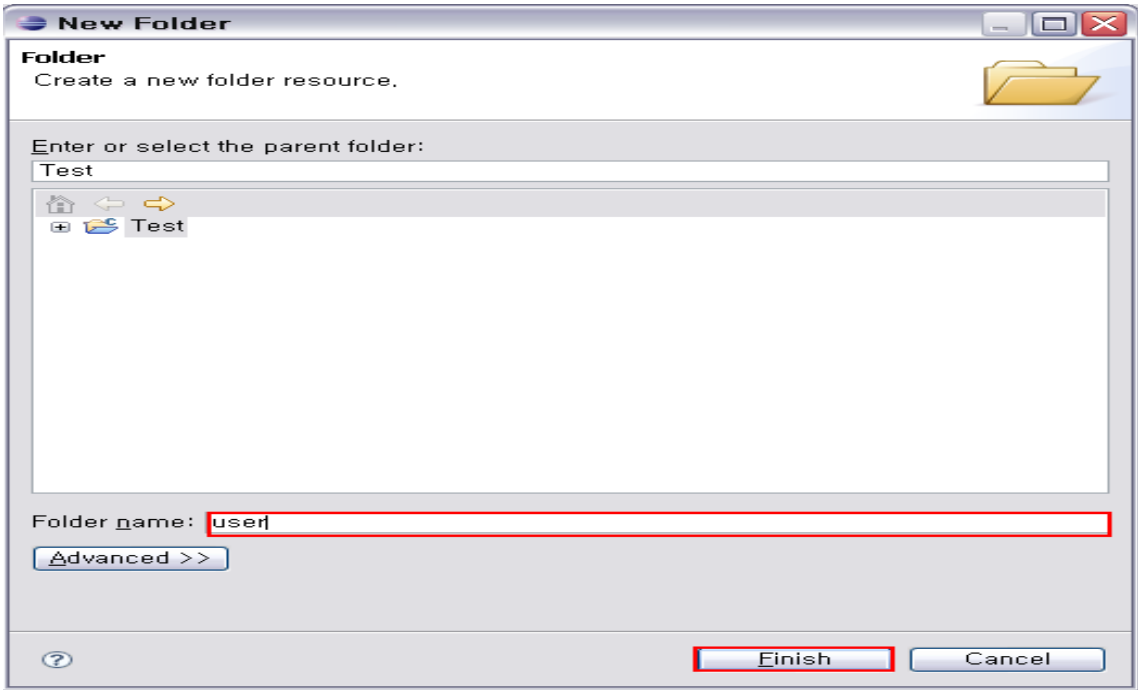
폴더(파일) 이름	기능
Libraries	STM32 펌웨어 라이브러리 API 폴더
System	인터럽트 핸들러 및 시스템 함수 폴더
Makefile	컴파일 수행 절차를 나열한 컴파일 스크립트 파일
stm32_flash.ld	STM32 메모리 영역 설정(Stack, heap 등을 설정하는 링커 스크립트) 파일

프로젝트 기본 설정을 마무리되었으므로, 지금부터 소스 코드를 작성할 파일 및 폴더 생성방법에 대해서 알아보도록 하자.

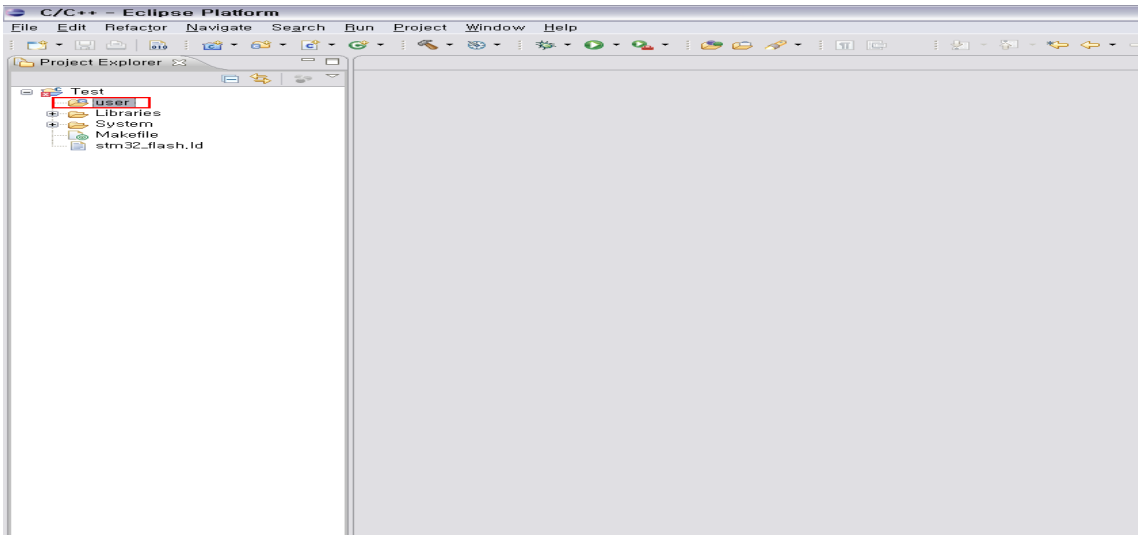
우선 소스 및 헤더 파일의 쉬운 관리를 위해 폴더를 생성하는 방법에 대해서 알아보도록 한다.

새로운 폴더를 생성하는 방법은 아래 그림과 같이 만들어 놓은 프로젝트 이름 위에서 마우스 오른쪽 버튼을 누르면 나타나는 메뉴를 통해서 생성한다.



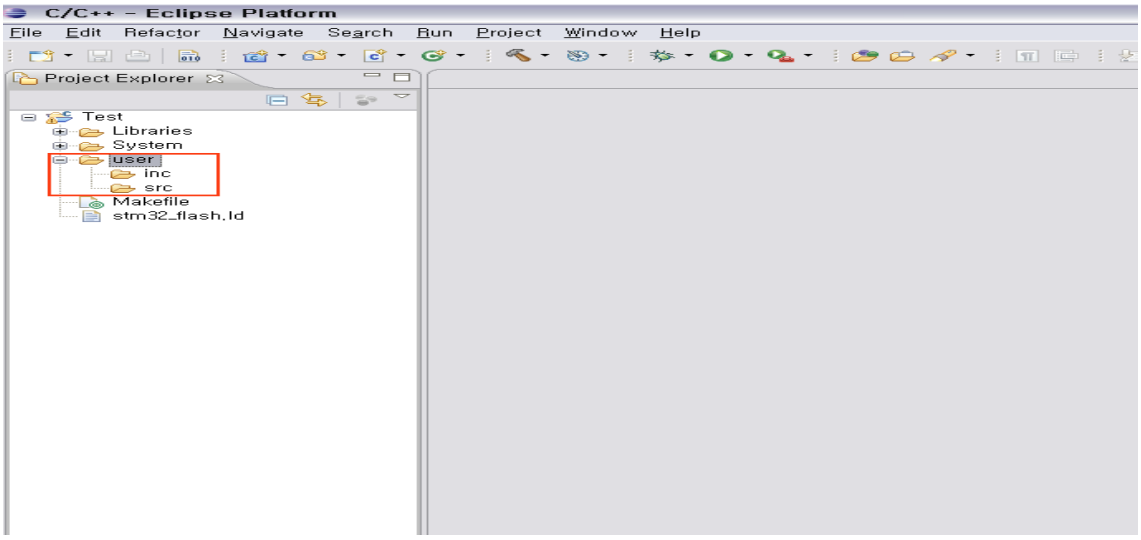


위 과정을 마치게 되면 아래 그림과 같이 만들어 놓은 프로젝트에 user이라는 폴더가 생성된 것을 확인할 수 있다.

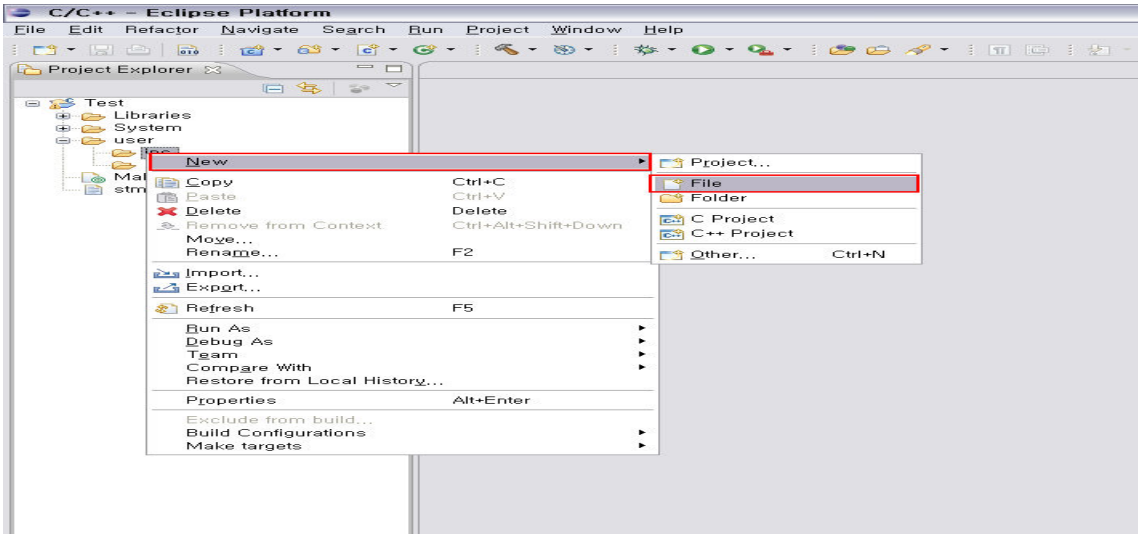


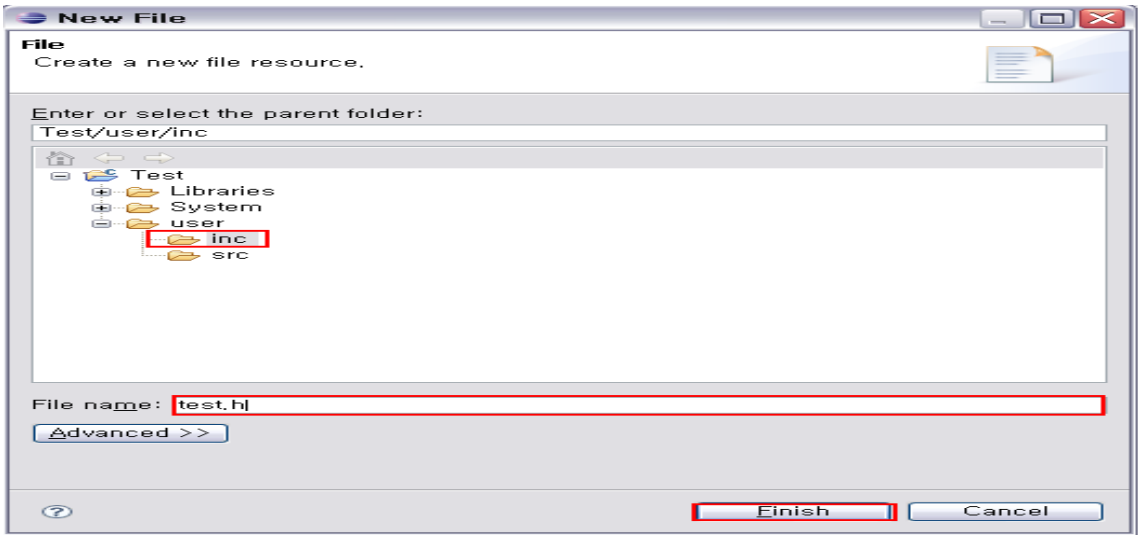
위와 같은 방법으로 test.c, test.h, main.c 파일을 프로젝트에 생성해보도록 한다. 각 파일들의 위치는 test.c, test.h 파일은 방금 추가한 user 폴더에 생성할 것이며, main.c 파일은 프로젝트 루트에 생성하도록 한다. 물론 아무 위치에 생성하더라도 드래그&드랍으로 이동이 가능하므로 실수를 하게 되더라도 크게 신경 쓰진 않아도 된다. 외부 파일을 추가하고자 하는 경우는 앞에서 기본 파일들을 추가했던 방식으로 추가를 하면 쉽게 추가할 수 있다. 물론 드래그&드랍으로도 가능하다.

폴더 구성을 좀 더 세분화하기위해서 user 폴더 내부에 inc와 src 폴더를 생성해서 작성할 파일들을 소스 파일과 헤더 파일로 좀 더 세분화하는 것도 좋은 방법이 된다.

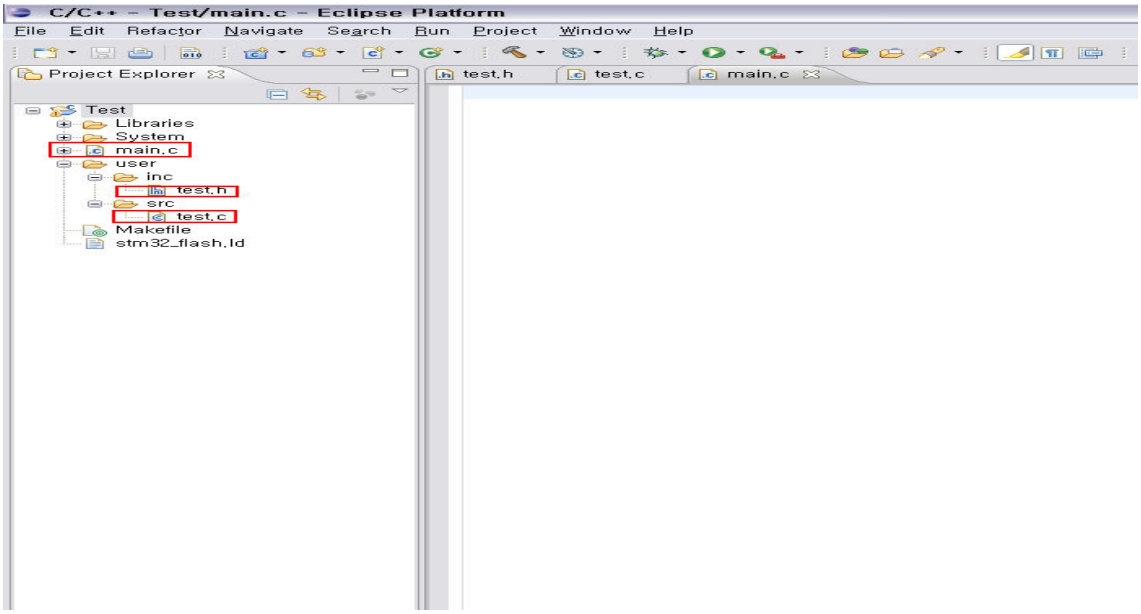


소스 파일들을 위치시킬 폴더가 구성 되었다면 이제 파일을 추가해보도록 한다. 파일을 추가하는 방법도 폴더를 생성했던 방법과 거의 같기 때문에 쉽게 추가가 가능하다.





같은 방법으로 test.c 파일과 main.c 파일도 추가하도록 한다. main.c 파일의 경우 컴파일 설정에서 프로젝트 루트 위치에서 찾도록 되어져 있기 때문에 main.c 파일은 프로젝트 루트에 추가하거나 다른 폴더에 생성 후 이동시키도록 한다. 물론 Makefile을 변경하여 위치를 정하면 되지만 가급적이면 Makefile의 수정은 하지 않는 방향으로 진행하도록 한다. 모든 파일을 추가했다면 아래와 같은 구성이 될 것이다.



다음으로 LED On/Off 코드를 추가하여 컴파일 과정을 수행해보자.

test.h

```
#ifndef __TEST_H__
#define __TEST_H__
void Led_On(void);
void Led_Off(void);
#endif
```

test.c

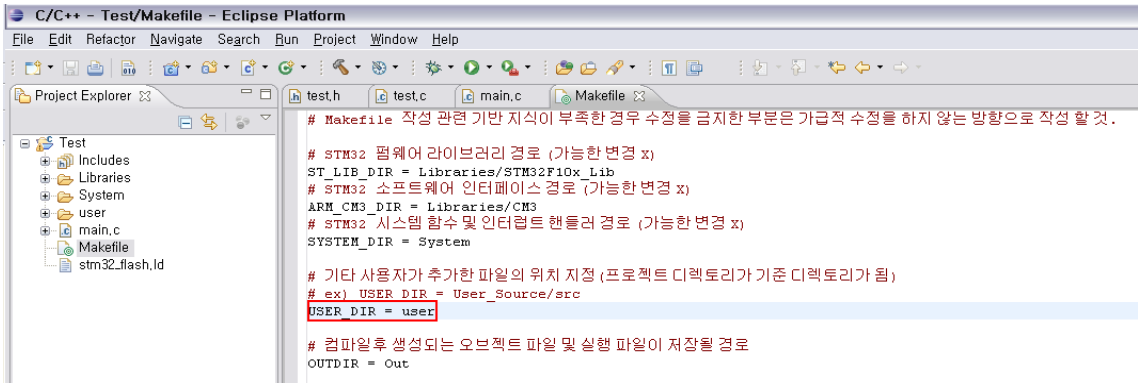
```
#include "stm32f10x.h"
#include "test.h"

void Led_On(void)
{
        GPIO_SetBits(GPIOD, GPIO_Pin_7);
        GPIO_SetBits(GPIOD, GPIO_Pin_13);
        GPIO_SetBits(GPIOD, GPIO_Pin_14);
        GPIO_SetBits(GPIOD, GPIO_Pin_15);
}

void Led_Off(void)
```

```
{
        GPIO_ResetBits(GPIOD, GPIO_Pin_7);
        GPIO_ResetBits(GPIOD, GPIO_Pin_13);
        GPIO_ResetBits(GPIOD, GPIO_Pin_14);
        GPIO_ResetBits(GPIOD, GPIO_Pin_15);
}
```

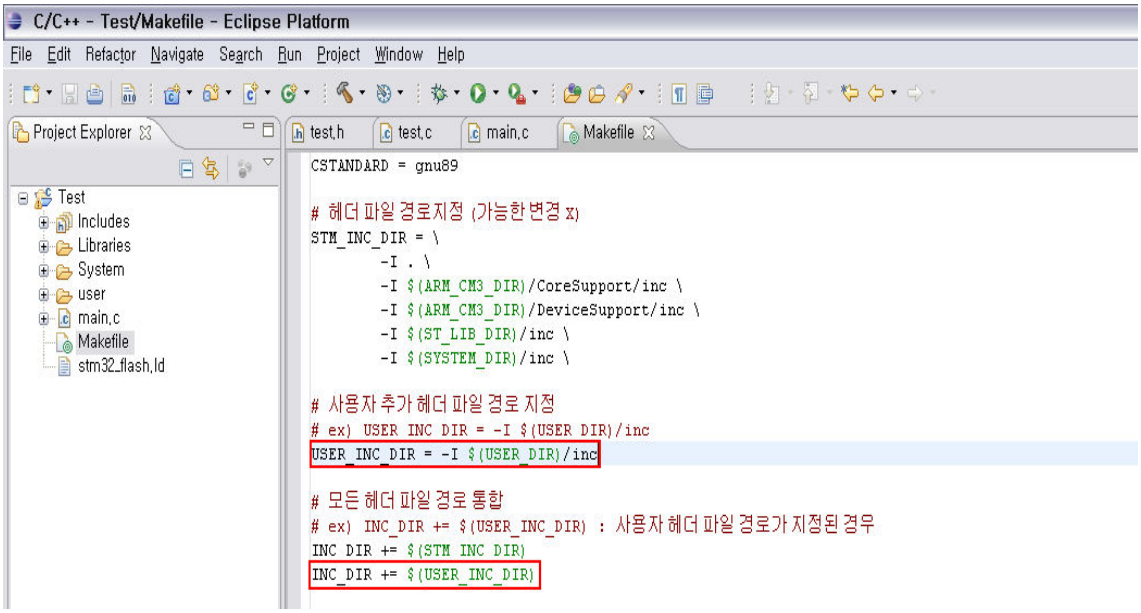
main.c

```
#include "test.h"
#include "stm32f10x.h"

int main(void)
{
        int x = 0;
        GPIO_InitTypeDef GPIO_InitStructure;
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_Init(GPIOD, &GPIO_InitStructure);

        while(1)
        {
                Led_On();
                for(x = 0; x < 0xFFFFF; x++);
                Led_Off();
                for(x = 0; x < 0xFFFFF; x++);
        }
        return 0;
}
```

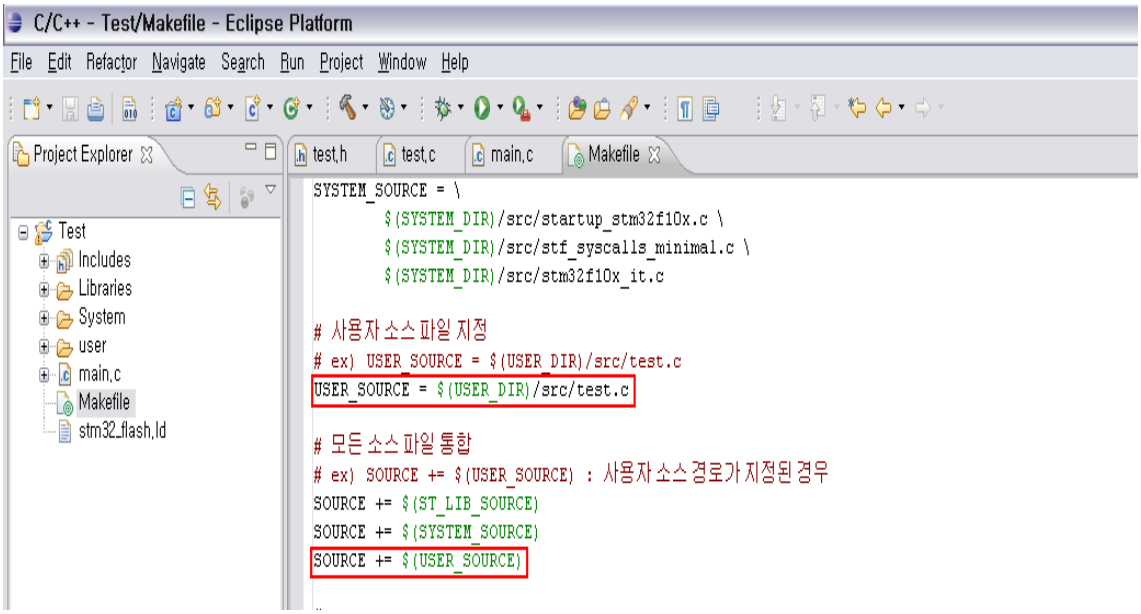
코드 작성이 완료되었다면, 프로젝트 루트에 있는 Makefile 파일에 조금 전에 새롭게 작성하여 추가한 소스 파일을 추가하도록 한다. USER_DIR에 test.c test.h 파일이 존재하는 폴더를 명시한다.(USER_DIR은 일종의 변수라고 생각하면 된다.)



USER_INC_DIR에 test.h 파일이 존재하는 폴더를 명시하고, INC_DIR에 USER_INC_DIR을 추가함으로써 헤더 파일의 존재를 컴파일러에 알린다. (-I 는 "Include"를 의미하는 컴파일 옵션이다.)

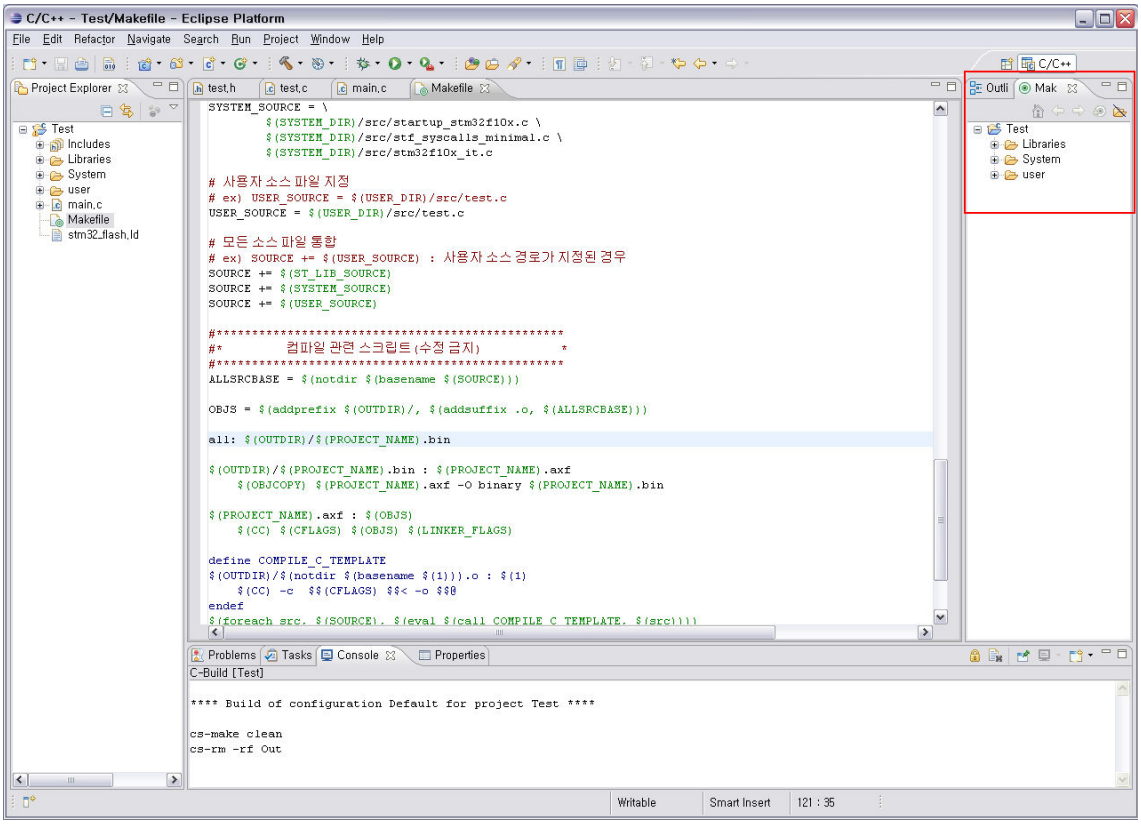


USER_SOURCE에 test.c 파일의 전체 경로를 명시하고, SOURCE에 USER_SOURCE를 추가함으로써 소스 파일의 존재를 컴파일러에 알린다.

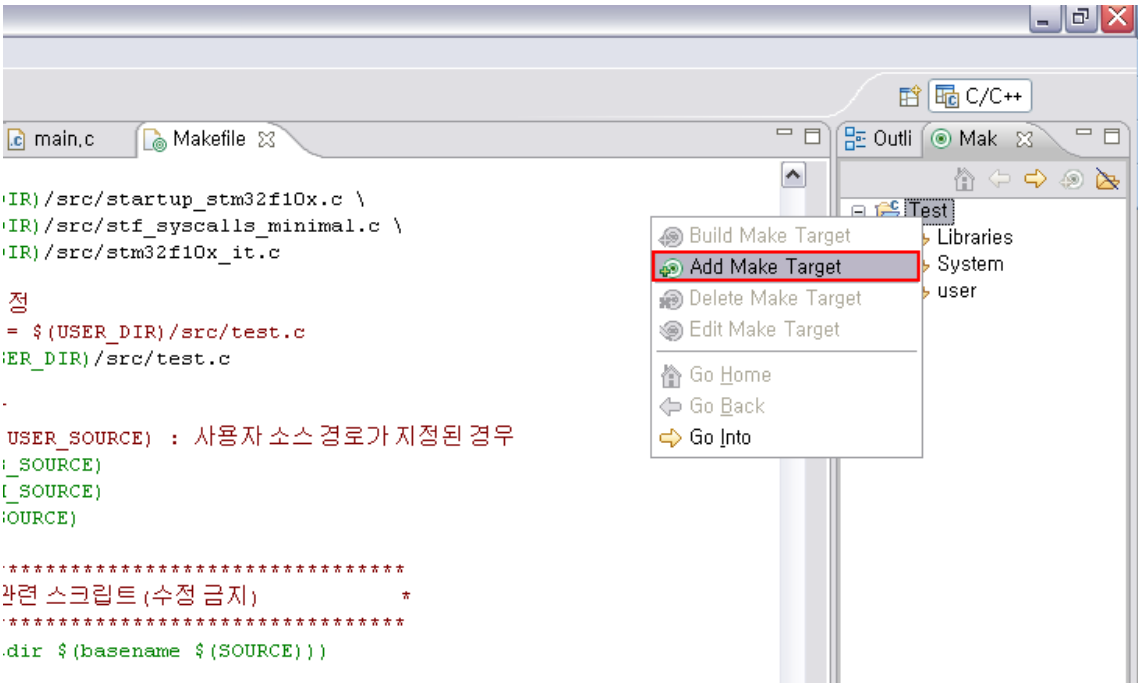


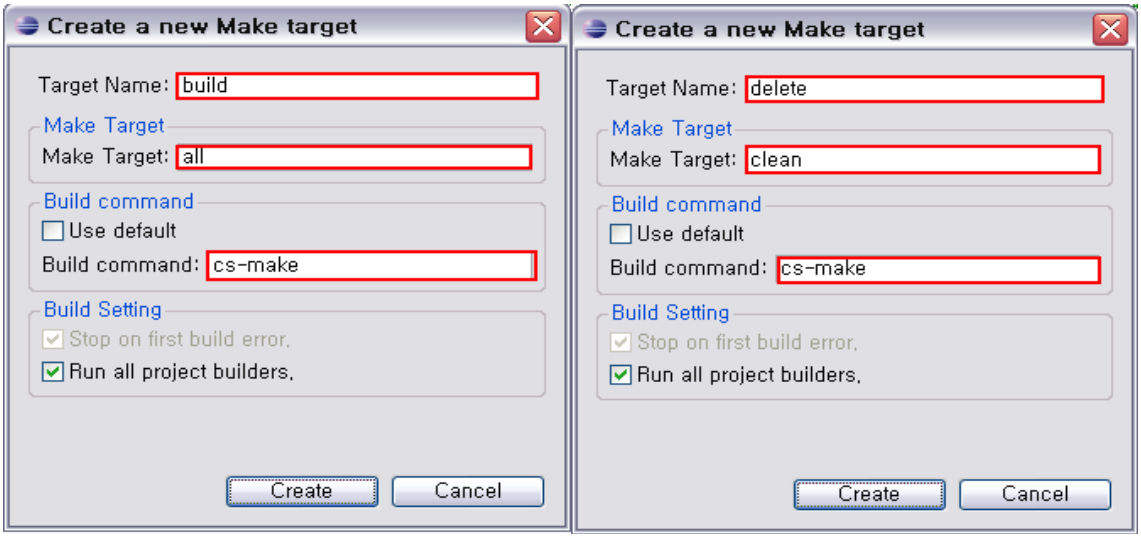
추가한 폴더나 파일이 여럿일 경우 위와 동일하게 연속적으로 추가하면 된다. 경로 및 파일 설정이 마무리 되었다면 컴파일러를 동작시키기 위한 명령을 등록하도록 한다. Eclipse 화면에서 우측에 Outline 탭과 Make Targets 탭이 존재한다. 여기서 Make Targets 탭에 컴파일 명령을 등록하고 더블 클릭으로 컴파일러를 동작시킬 수 있다.

아래 Eclipse에서 Make Targets 탭을 클릭해보면 지금까지 구성한 프로젝트 폴더의 구조로 폴더들이 보일 것이다. 여기서 중요한 부분은 Makefile이라는 이름을 가진 파일이 어느 폴더에 위치하는가가 중요한 부분이 되며, 이 Makefile이 존재하는 폴더에 컴파일러 실행 명령을 추가하여야만 컴파일이 가능하다는 것을 명심해야한다.



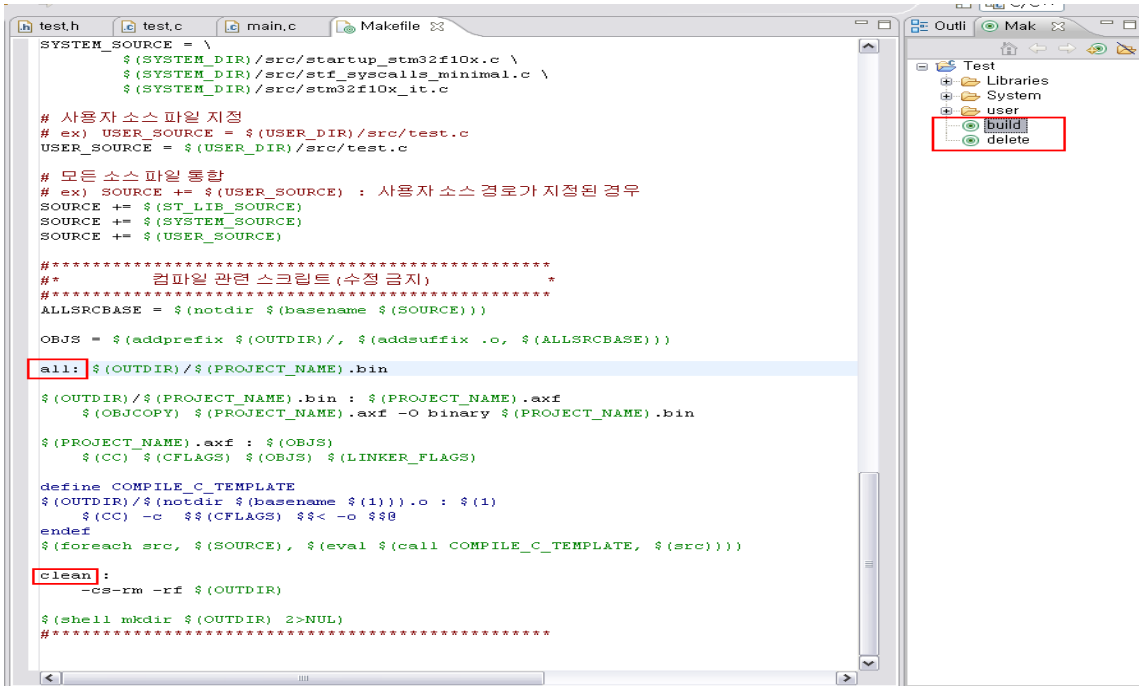
프로젝트 구조에서 Makefile은 Test 폴더에 존재한다. 따라서 Make Targets 부분에서도 Test에 컴파일러 실행 명령을 등록해야한다. 등록 방법은 Make Targets 탭의 Test 폴더에서 오른쪽 마우스를 누르면 나타나는 메뉴를 통해서 가능하며, 아래 그림과 같이 진행하면 된다.





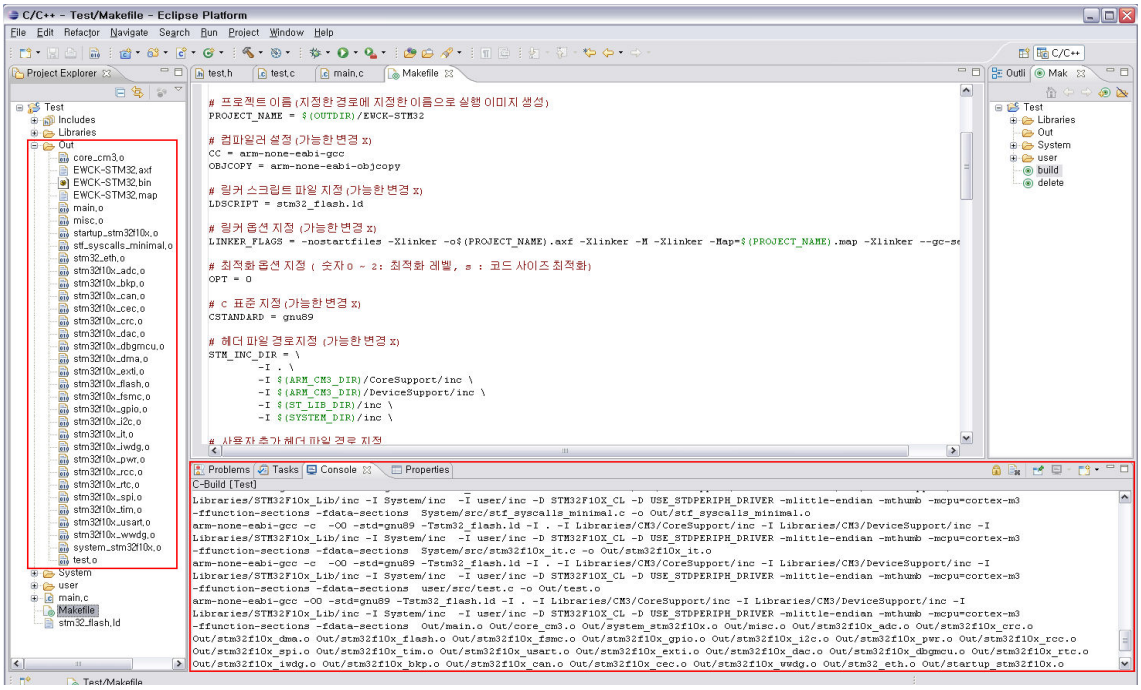
항목	기능
Target Name	실행 명령 항목의 이름 설정
Make Target	Makefile에서 실행할 위치 지정
Build command	Makefile를 실행시킬 실행 파일 이름 지정

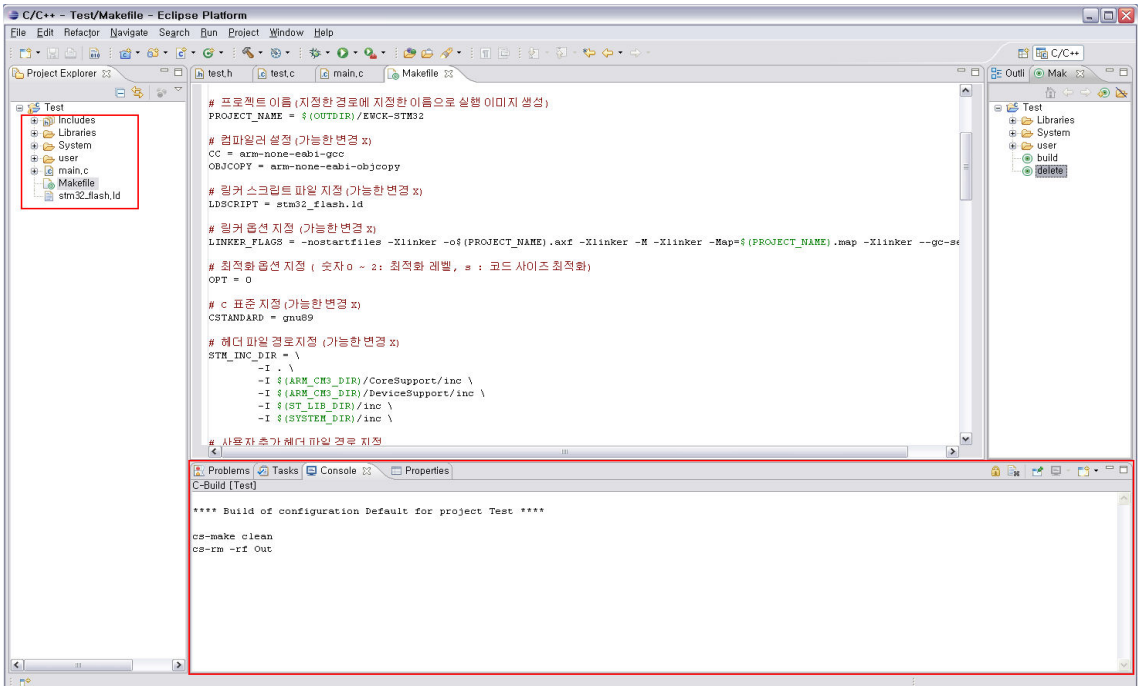
Make Targets 부분은 위 그림과 같이 작성한다고 해서 무조건 실행되어 지는 것은 아니다. 위 그림의 all이나 clean이 Makefile에 존재해야지만 사용할 수 있는 부분이다. Makefile에서 이런 부분을 Target라고 부르며 특정 명령어들을 수행시키는 머리말 부분이 Target라고 생각하면 된다.



실행 항목을 추가하게 되면 위 그림과 같이 build와 delete 항목이 추가된 것을 확인할 수 있으며, 위에서 언급했듯이 Makefile 내부에 all과 clean이 존재하는 것도 확인할 수 있다.

위에서 추가한 실행 항목 중 build를 더블 클릭해 실행해보면 Eclipse 하단에 있는 Console 탭에 컴파일 과정이 출력되며, 컴파일 후 생성되는 파일들은 Makefile에서 지정한 Out 폴더에 생성이 되게 된다. 만약 컴파일 결과 생성되는 파일들을 삭제하고 싶다면 추가한 실행 항목 중 delete를 더블 클릭해 삭제가 가능하다.



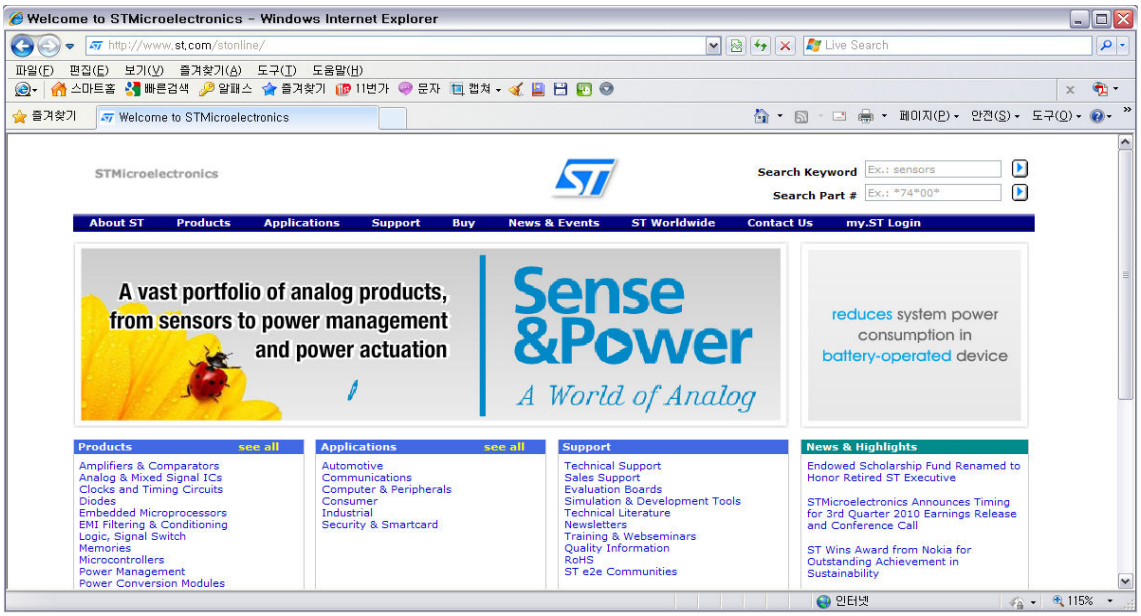


컴파일 후 생성되는 EWCK-STM32.bin 파일을 보드에 다운로드하면 LED가 깜박이는 것을 확인할 수 있다.

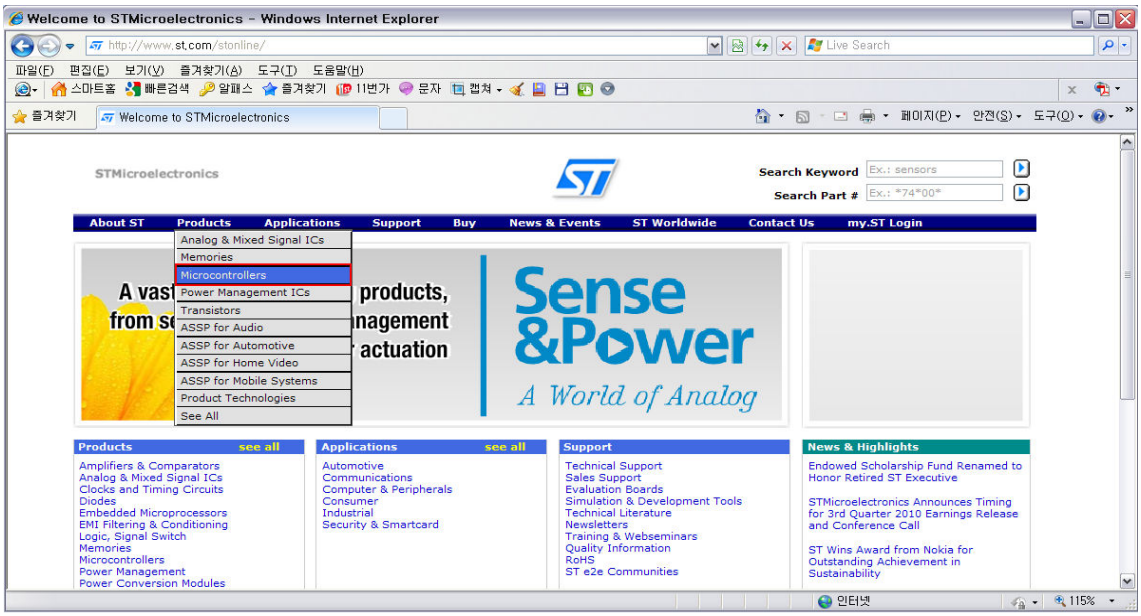
1.4 FlashLoader 설치하기

Flash Loader는 Serial 인터페이스를 이용해 타겟 시스템(보드)에 컴파일 결과로 생성되는 실행 파일 이미지(.bin)를 메모리에 적재시켜주는 프로그램이다. 이 프로그램은 ST 마이크로 홈페이지에서 다운로드 받을 수 있다.

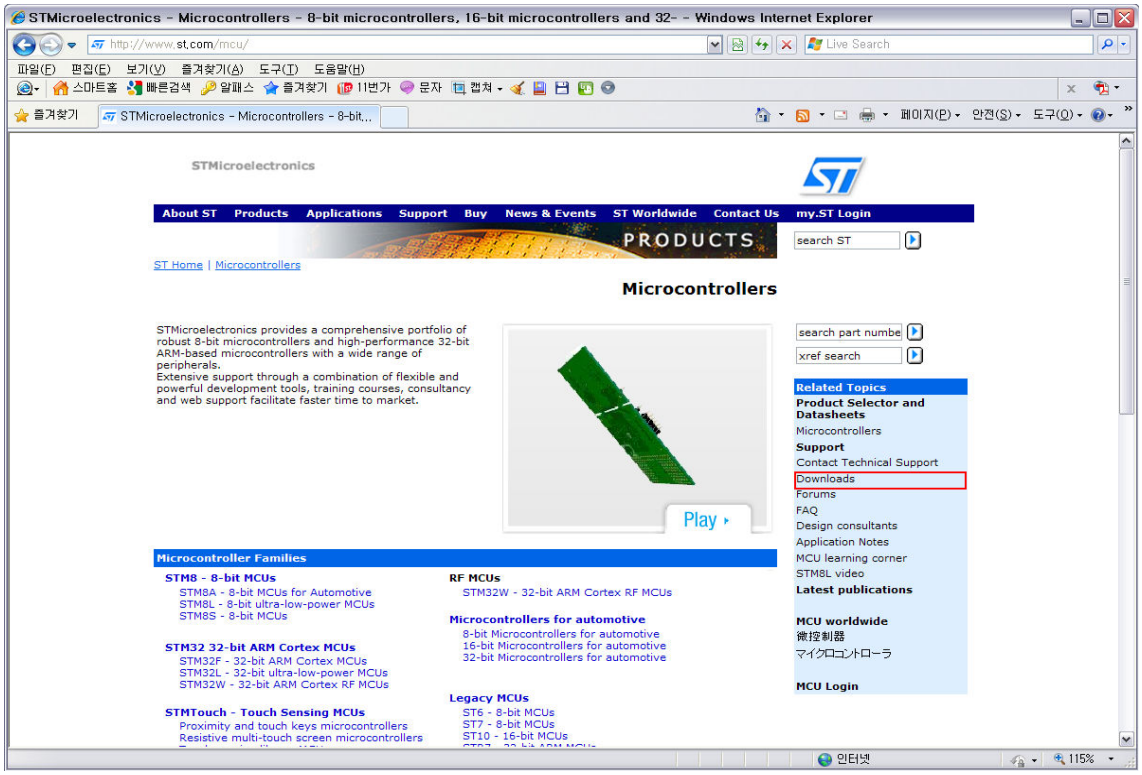
사이트 주소 : http://www.st.com/



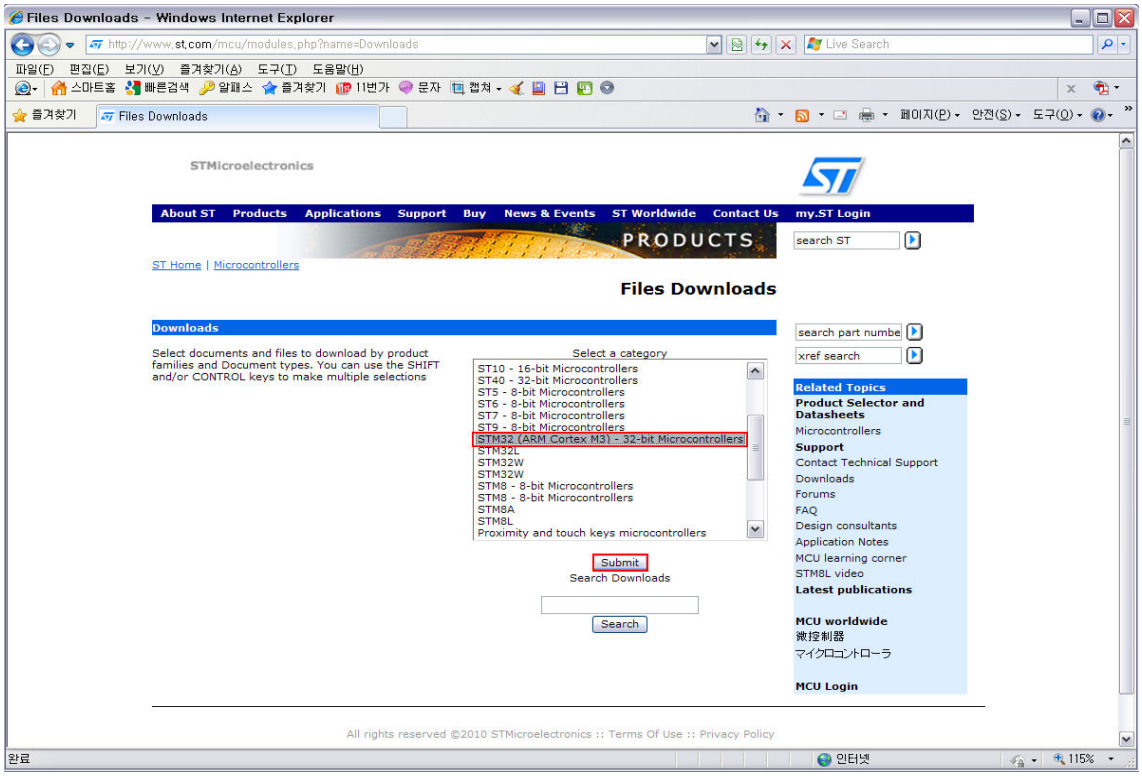
상단의 메뉴에서 Products 메뉴를 선택하고 Microcontrollers 메뉴를 선택한다.



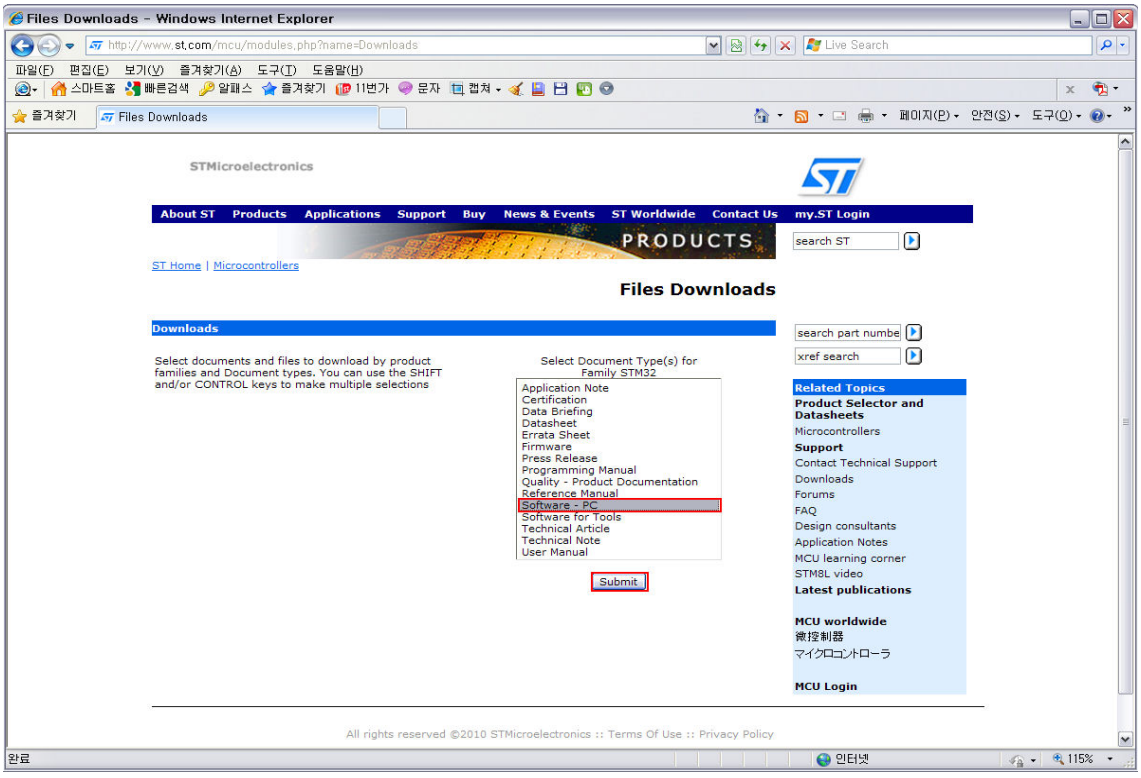
우측에 나타나는 서브 메뉴 중 Downloads를 선택한다.



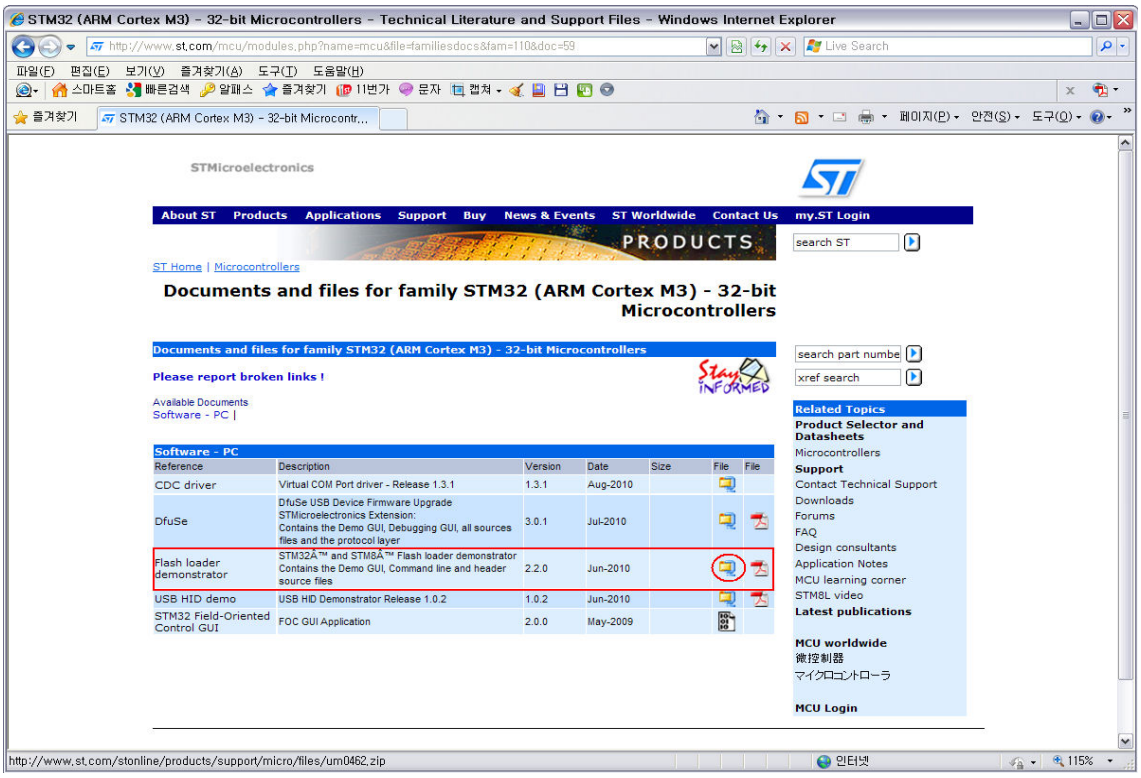
중앙에 나타나는 목록 중 STM32(ARM cortex M3) - 32-bit Microcontrollers를 선택하고 Submit 버튼을 누른다.



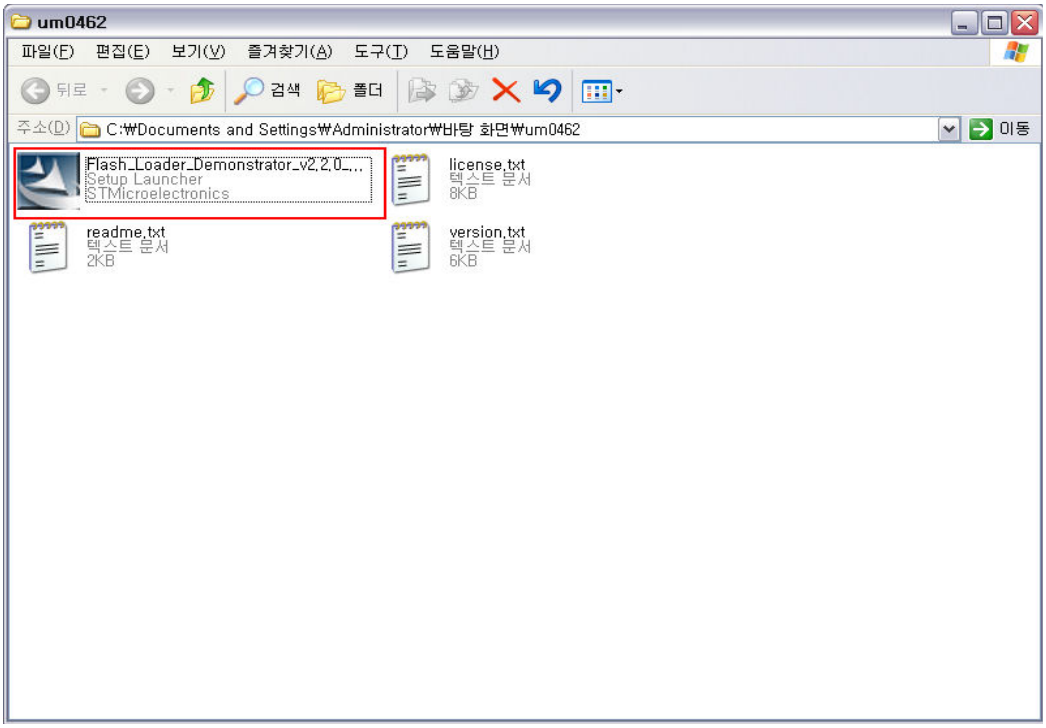
중앙에 나타난 목록 중 Software - PC를 선택하고 Submit를 선택한다.



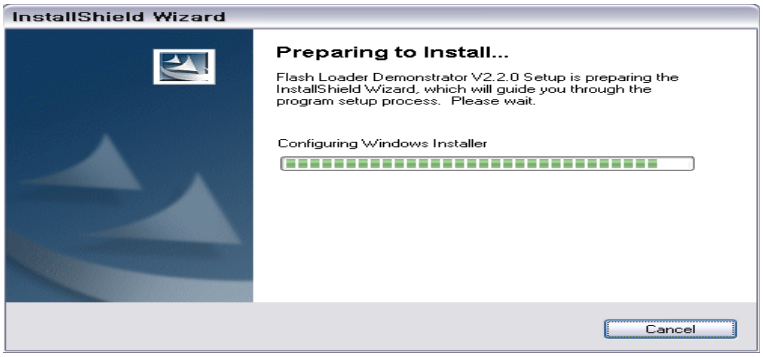
화면의 소프트웨어 목록 중 Flash loader demonstrator에 있는 압축 파일을 클릭하여 다운로드 받는다.

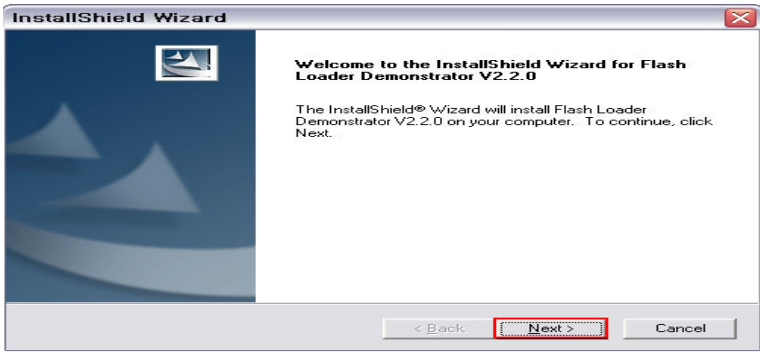


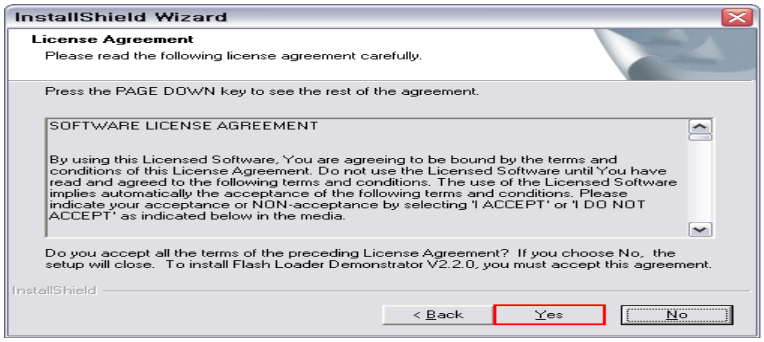
압축 파일의 이름은 UM0462.zip이며 압축을 풀면 Flash_Loader_Demonstrator_v2.2.0_Setup.exe 설치 파일이 존재하는 것을 확인할 수 있다.

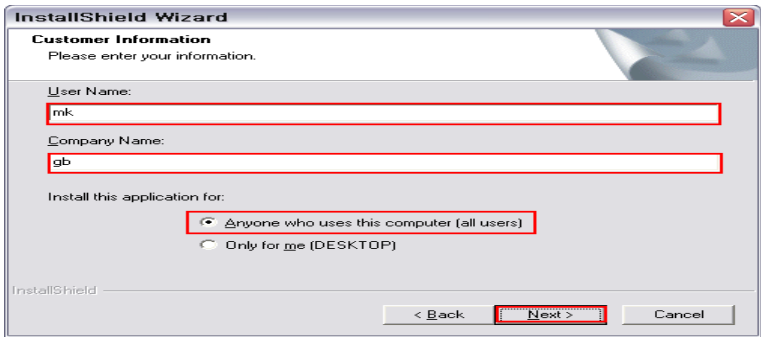


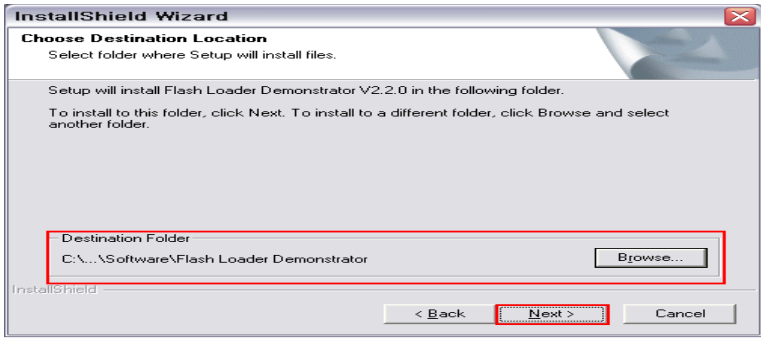
Flash Loader의 설치는 간단하게 실행 파일을 더블 클릭해서 실행하고 간단히 몇 가지 기본사항들만 기입하게 되면 설치를 마칠 수 있다.

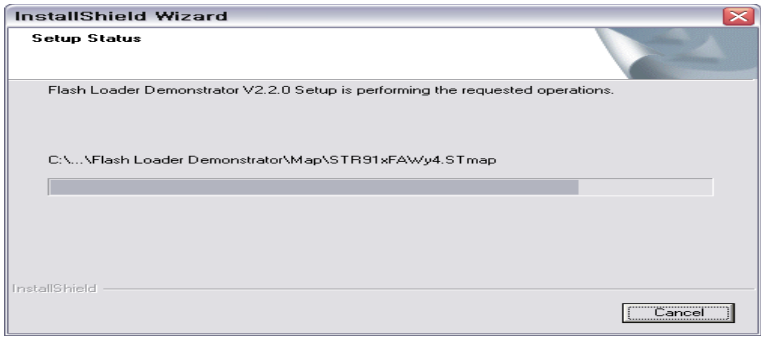


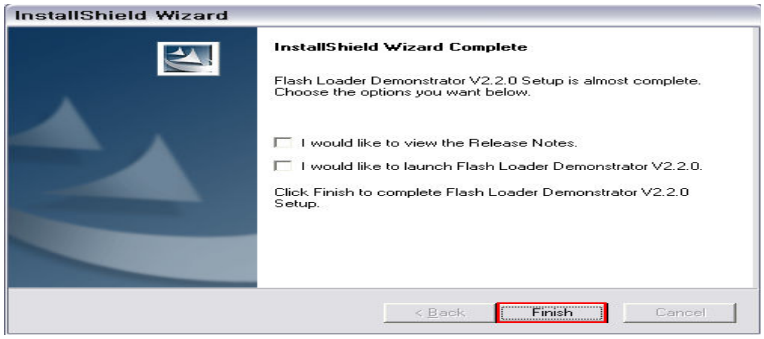



InstallShield Wizard
License Agreement
Please read the following license agreement carefully.
Press the PAGE DOWN key to see the rest of the agreement.
SOFTWARE LICENSE AGREEMENT
By using this Licensed Software, You are agreeing to be bound by the terms and conditions of this License Agreement. Do not use the Licensed Software until You have read and agreed to the following terms and conditions. The use of the Licensed Software implies automatically the acceptance of the following terms and conditions. Please indicate your acceptance or NON-acceptance by selecting 'I ACCEPT' or 'I DO NOT ACCEPT' as indicated below in the media.
Do you accept all the terms of the preceding License Agreement? If you choose No, the setup will close. To install Flash Loader Demonstrator V2.2.0, you must accept this agreement.
InstallShield
< Back
Yes
No

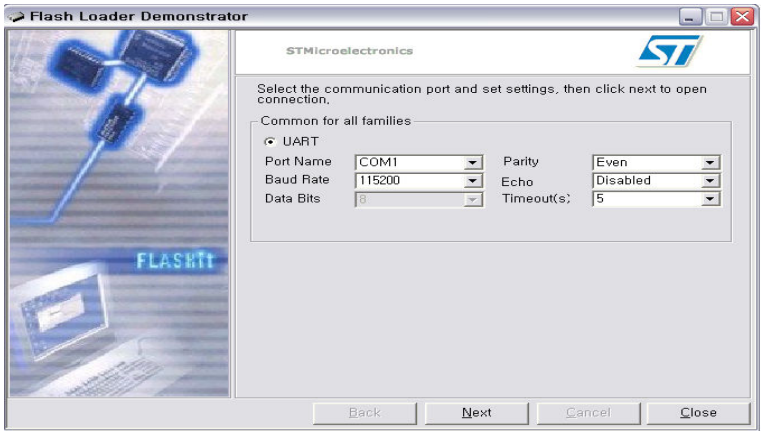

InstallShield Wizard
Customer Information
Please enter your information.
User Name:
mk
Company Name:
gb
Install this application for:
Anyone who uses this computer (all users)
Only for me (DESKTOP)
InstallShield
< Back
Next >
Cancel


InstallShield Wizard
Choose Destination Location
Select folder where Setup will install files.
Setup will install Flash Loader Demonstrator V2.2.0 in the following folder.
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.
Destination Folder
C:\...\Software\Flash Loader Demonstrator
Browse...
InstallShield
< Back
Next >
Cancel


InstallShield Wizard
Setup Status
Flash Loader Demonstrator V2.2.0 Setup is performing the requested operations.
C:\...\Flash Loader Demonstrator\Map\STR91xFAWy4.STmap
InstallShield
Cancel


InstallShield Wizard
InstallShield Wizard Complete
Flash Loader Demonstrator V2.2.0 Setup is almost complete. Choose the options you want below.
I would like to view the Release Notes.
I would like to launch Flash Loader Demonstrator V2.2.0.
Click Finish to complete Flash Loader Demonstrator V2.2.0 Setup.
< Back
Finish
Cancel

설치를 마무리하고 Flash Loader를 실행하게 되면 아래 그림과 같은 구성의 초기 화면으로 실행되게 된다.



Flash Loader의 설치가 마무리 되었다면 타겟 보드에 컴파일 후 생성된 EWCK-STM32.bin 파일을 다운로드해야지만 결과를 확인할 수 있는데, 실행 파일 다운로드를 위해서 타겟 보드의 점퍼를 변경해주어야만 가능하다.

EWCK-STM32 타겟 보드의 전원 단자 옆에 4개의 딥스위치로 구성된 점퍼가 존재하는 것을 볼 수 있는데, 이 점퍼를 어떻게 Setting 하는가에 따라 시스템 부팅 모드와 실행 모드로 동작 시킬 수 있게 된다. 각 모드에 대해서 간단하게 설명을 하자면, 우선 시스템 부팅 모드는 STM32 MCU에 이미 내장되어 있는 프로그램으로 MCU를 동작시키는 모드로 우리가 작성한 실행 파일을 다운로드 할 수 있도록 해주는 모드이다. 또 하나의 모드인 실행 모드는 우리가 작성한 프로그램으로 MCU를 동작시키는 모드이다.

따라서 새로운 프로그램을 다운로드하기 위해서는 시스템 부팅 모드로 점퍼를 변경하여야하며, 새로운 프로그램으로 보드를 동작하게 하려면 실행 모드로 점퍼를 변경하면 된다. 점퍼는 아래와 같이 변경하여 두 모드로 변경이 가능하다.

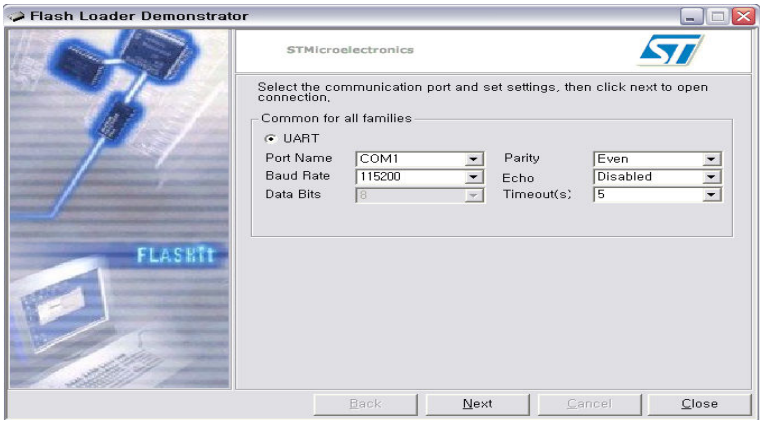
시스템 부팅 모드

1번	2번	3번	4번
	on		on
off		off	

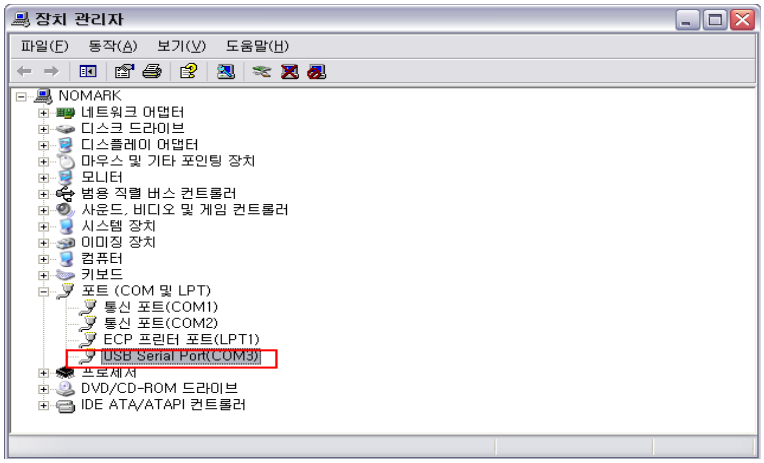
실행 모드

1번	2번	3번	4번
on		on	
	off		off

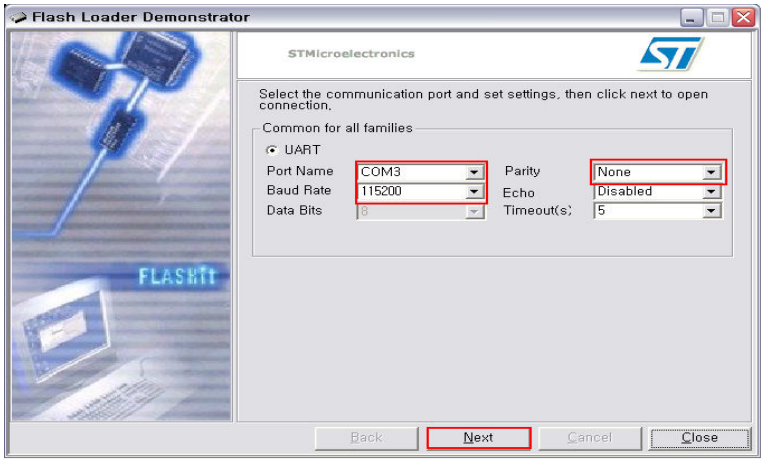
이전에 작성해 놓은 LED 프로그램을 다운로드하기 위해서 보드의 부팅 모드를 시스템 부팅 모드로 변경하고 전원을 인가하도록 한다. 참고로 보드에 전원이 인가된 상태에서 점퍼를 변경한 경우에는 반드시 전원을 껐다 다시 켜야지만 해당 모드로 동작이 이루어진다는 것을 기억하도록 한다. 다음으로 보드와 함께 제공되는 USB to Serial 케이블을 보드와 PC에 연결하고 Flash Loader 프로그램을 실행시키도록 한다.



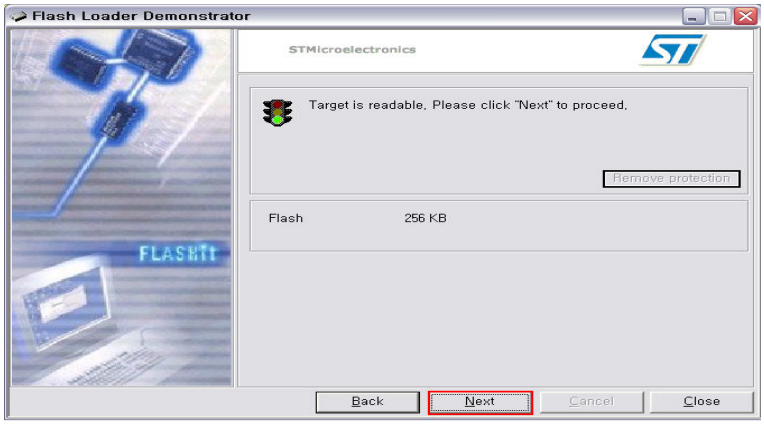
Flash Loader 프로그램에서 Port Name는 USB to Serial의 COM 포트 번호를 선택하고 Baud Rate는 115200, Parity는 None으로 설정하도록 한다. 자신의 COM Port번호를 모르는 경우 내 컴퓨터에서 오른쪽 마우스를 눌러 속성을 선택하고, 포트(COM 및 LPT)를 눌러 USB to Serial 부분의 포트 번호를 확인하고 선택하도록 한다.



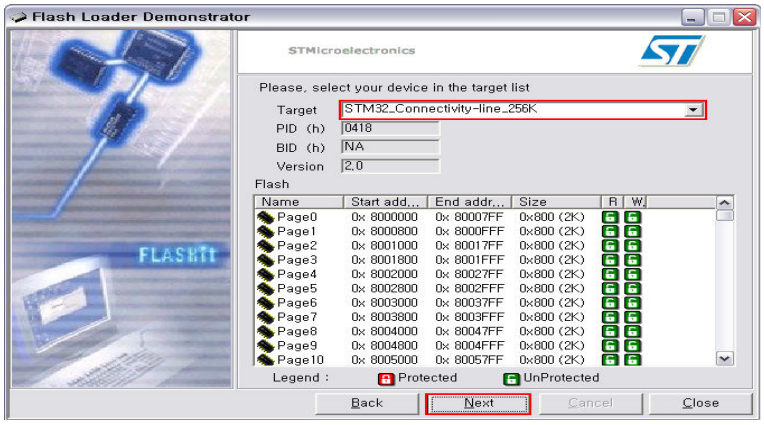
위에서 언급한바와 같이 설정을 하고 Next 버튼을 눌러 다음으로 이동하도록 한다.



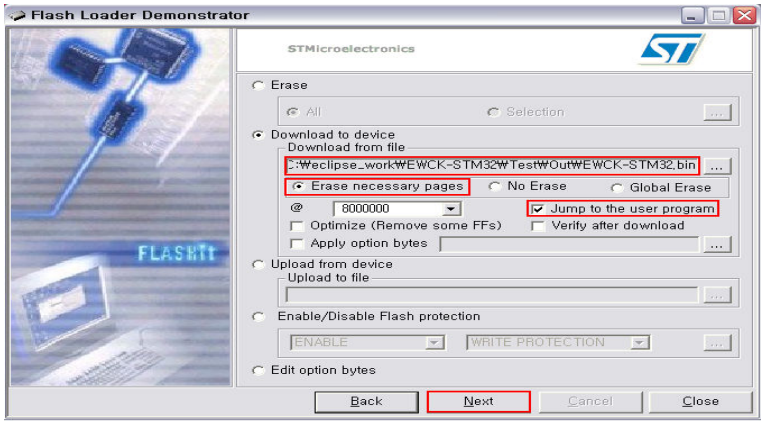
만약 아래와 같은 화면으로 표시되지 않고 경고 창이 나타날 경우는 프로그램과 보드간의 연결이 되지 않은 상태로, 보드에 전원이 인가되지 않았거나, Serial 케이블 연결 불량 또는 점퍼 설정이 정확하지 않은 경우이므로 위 사항을 확인하고 전원을 껐다 켠 후 Next 버튼을 눌러 재시도하기 바란다.

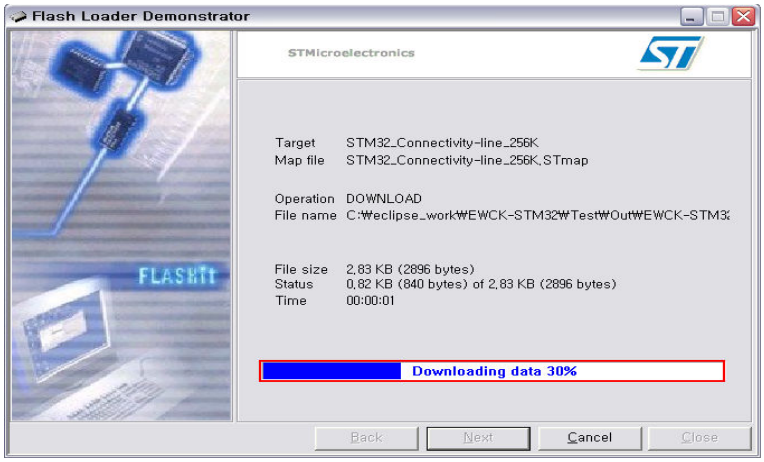


STM32 MCU에 대한 제품군 및 메모리 구조에 대한 내용을 보여주는 화면으로, EWCK-STM32는 STM32F107VCT6 MCU를 사용하고 있으며 Flash Memory 256KB, SRAM 64KB의 Connectivity Line 제품군에 해당한다.



다운로드 파일은 이전에 컴파일 해 놓은 EWCK-STM32.bin 파일로 선택하고, 프로그램이 써질 공간만 삭제한다. Jump to the User Program을 선택하고 업로드하게 되면 점퍼를 실행 모드로 변경하지 않고도 바로 사용자 프로그램을 실행하도록 할 수 있다. 단 프로그램 업로드가 완료되는 시점에 한해서만 실행이 가능하다. 점퍼 변경 없이 동작을 확인할 경우 사용하면 편리하다.





다운로드가 끝나게 되면 타겟 보드의 LED 4개가 깜박거리는 것을 확인할 수 있다.

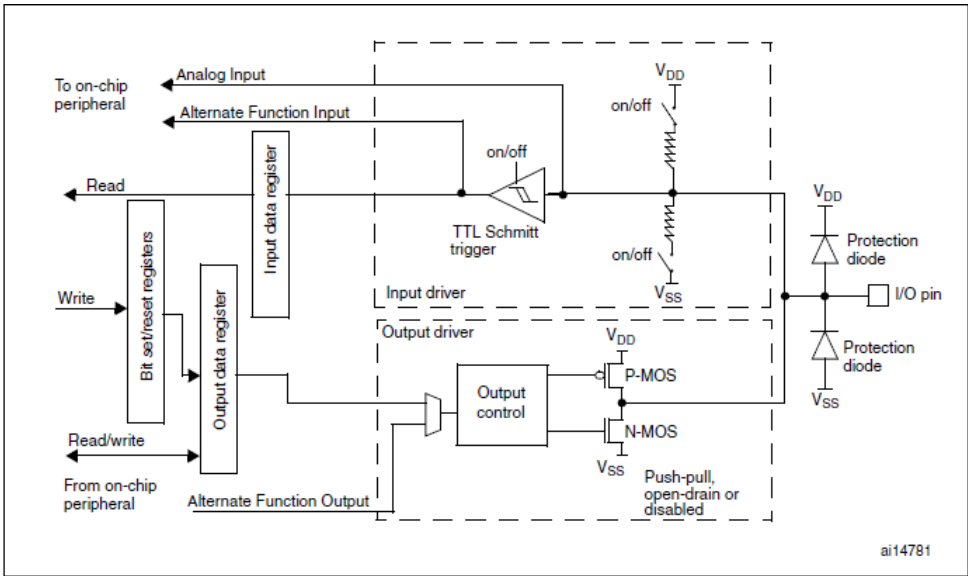
실습 2. GPIO 제어하기

2.1 GPIO (General Purpose Input/Output) 개요

GPIO란? General Purpose Input/Output의 줄임말로서 사용 목적이 정해져 있지 않으며, 필요에 따라 입력 또는 출력으로도 사용되어 지며, 그 외 Chip 자체 기능으로 사용이 가능한 Chip이 가지고 있는 하나하나의 핀을 이야기한다.

실습 보드에 사용된 chip은 STM32f107vc로 GPIOA~E 5개의 GPIO Block을 가지며, 하나의 Block마다 16개의 pin을 가지므로 총 80개의 GPIO pin을 가진다.

다음은 GPIO I/O 포트 비트의 기본 구조를 나타낸 것이다.



각 GPIO 포트는 두 개의 32bit Configuration 레지스터(GPIOx_CRH, GPIOx_GRL), 두 개의 32bit Data 레지스터(GPIOx_IDR, GPIOx_ODR), 한 개의 32bit Set/Reset 레지스터(GPIOx_BSRR), 한 개의 16bit Reset 레지스터(GPIOx_BRR)와 하나의 32bit Locking 레지스터(GPIOx_LCKR)를 가지고 있다.

다음은 각 포트를 위한 메모리 영역을 나타낸 것이다.

Boundary address	Peripheral
0x4001 1800 - 0x4001 1BFF	GPIO Port E
0x4001 1400 - 0x4001 17FF	GPIO Port D
0x4001 1000 - 0x4001 13FF	GPIO Port C
0x4001 0C00 - 0x4001 0FFF	GPIO Port B
0x4001 0800 - 0x4001 0BFF	GPIO Port A

2.2 GPIO 레지스터

GPIO 제어를 위해 먼저 설정해야 할 레지스터는 Port Configuration Register이다. Port Configuration Register는 2가지 종류가 있다.

1. GPIOx_CRL(Port Configuration Register Low) (x=A..G)

Address offset : 0x00, Reset Value : 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

2. GPIOx_CRH (Port Configuration Register High) (x=A..G)

Address offset : 0x04, Reset Value : 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_CRL, GPIOx_CRH로 두 개의 레지스터가 있는데, 여기서 x=A..G로 표시가 되어 있는 것은 GPIO Port A부터 G를 의미하는 것이다. 레지스터의 주소 값은 서로 다르겠지만 GPIO Port A부터 G까지 해당 레지스터의 동작은 동일하므로 한꺼번에 설명하도록 한다.

위 내용에서 Address offset이 의미하는 것은 GPIO Port A부터 G까지 각각의 Base Address에서 Address offset만큼을 더한 것이 해당 레지스터의 주소 값이라는 것을 나타내 주고 있는 것이다. Address에 관한 자세한 내용은 STM32f107vc의 데이터 시트를 참조하도록 한다.

GPIOx_CRL는 Port x의 0~7번 pin의 설정을 담당하고, GPIOx_CRH는 8~15번 pin의 설정을 담당한다. 각 레지스터는 각 pin 별로 4bit 씩 설정할 수 있도록 되어 있다. 즉, GPIOx_CRL의 0~3번 bit는 Port x의 0번 pin의 설정을 담당한다. 이 레지스터에서 설정할 수 있는 것은 CNF와 MODE이다.

우선 MODE에 관한 내용부터 살펴보자.

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0	**MODEy[1:0]:** Port x mode bits (y= 0 .. 7) These bits are written by software to configure the corresponding I/O port. 00: Input mode (reset state) 01: Output mode, max speed 10 MHz. 10: Output mode, max speed 2 MHz. 11: Output mode, max speed 50 MHz.

GPIOx_CRL, CRH의 초기 값이 0x4444 4444이므로 한 pin에 대한 4bit의 초기 값은 0100 인 것을 알 수 있다. 즉, MODE의 초기 값은 00으로 Input mode 로 동작하게 된다. 01~11까지는 Output mode에 대한 설정과 주파수 설정을 위한 값이다.

다음은 CNF 설정에 대한 내용을 살펴보자.

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2	**CNFy[1:0]:** Port x configuration bits (y= 0 .. 7) These bits are written by software to configure the corresponding I/O port. **In input mode (MODE[1:0]=00):** 00: Analog mode 01: Floating input (reset state) 10: Input with pull-up / pull-down 11: Reserved **In output mode (MODE[1:0] > 00):** 00: General purpose output push-pull 01: General purpose output Open-drain 10: Alternate function output Push-pull 11: Alternate function output Open-drain

현재 MODE의 설정이 Input인지 Output인지에 따라 구분된다. 초기 값이 '0100' 이므로 Input mode에서 Floating input 모드가 초기 설정 값이다. Analog mode는 말 그대로 디지털 값을 입력 받는 것이 아니라 변화하는 아날로그 신호를 그대로 받는 것이다. pull-up/ pull-down mode는 내부에 pull-up/ pull-down이 자동으로 걸려있는 상태에서 입력을 받게 되는 모드를 뜻하며, Floating input은 반대로 내부에 pull-up/ pull-down이 걸려있지 않고 외부에서 사용자가 마음대로 변경시켜 사용하는 모드를 의미한다.

Output mode일 경우 General purpose을 사용할 것인지 Alternate function을 사용할 것인지와 push-pull 로 할 것인지 Open-drain으로 할 것인지를 설정한다. Alternate function은 각 pin의 Remap 설정에 따라 I/O Port 외의 기능 핀으로 사용할 수 있는 것으로, 예를 들면 다음 표에서 볼 수 있듯이 PE11 pin의 경우 TIM1_CH2번 pin, 즉 2번 타이머로도 사용될 수 있다.

Alternate functions mapping	TIM1_REMAP[1:0] = "00" (no remap)	TIM1_REMAP[1:0] = "01" (partial remap)	TIM1_REMAP[1:0] = "11" (full remap) (1)
TIM1_ETR	PA12		PE7
TIM1_CH1	PA8		PE9
TIM1_CH2	PA9		PE11
TIM1_CH3	PA10		PE13
TIM1_CH4	PA11		PE14

자세한 내용은 데이터 시트를 참조하도록 한다.

3. GPIOx_IDR (Port Input Data Register)

Address offset : 0x08, Reset Value : 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

GPIOx Port로 입력되는 각 bit가 저장되는 레지스터이다.

4. GPIOx_ODR (Port Output Data Register)

Address offset : 0x0C, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx Port로 출력되는 각 bit가 저장된다. GPIOx_BSRR 레지스터에 의해 각 bit가 Set되거나 Clear 될 수 있다.

5. GPIOx_BSRR (Port Bit Set/Reset Register)

Address offset : 0x10, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

레지스터의 이름이 Set Register가 아닌 Set/Reset Register인 이유는 GPIOx_BSRR으로 Set과 Reset을 모두 처리할 수 있기 때문이다. 상위의 31:16번 bit가 BRy bit로 해당 pin을 Reset 시키며, 하위의 15:0 bit인 BSy bit로 해당 pin을 Set 할 수 있다. 만약 BSy와 BRy가 동시에 '1' 로 쓰여 질 경우 BSy가 더 높은 우선순위를 가진다. GPIOx_BRR 레지스터가 따로 존재하므로 GPIOx_BSRR 레지스터는 보통 해당 pin을 Set하는 경우에만 사용된다.

6. GPIOx_BRR (Port Bit Reset Register)

Address offset : 0x14, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

하위 16bit 각각의 값이 하나의 pin을 Reset 시킨다. 원하는 pin에 해당하는 bit에 '1' 을 써 넣으면 GPIOx_ODR(Output Data Register)의 해당 bit 값이 '0' 으로 Clear 된다.

7. RCC_APB2ENR (APB2 Peripheral Clock Enable Register)

Address offset : 0x18, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

Port를 사용하기 위해서 각 포트의 Clock을 Enable 시킨다. '0' 이 설정되면 Disable, '1' 이 설정되면 Clock이 Enable 된다.

2.3 GPIO 라이브러리 함수

```
(*(volatile unsigned long *)0x40011400) &= ~(0xF0000000);  /* GPIOD_CRL 31:28 bit Clear */
(*(volatile unsigned long *)0x40011400) |= 0x30000000;     /* GPIOD_CRL 31:28 bit Set 0011 */
(*(volatile unsigned long *)0x40011410) |= 0x0080;         /* GPIOD_BSRR 7 bit Set */
```

위 소스는 LED를 제어하는 소스의 일부분이다. GPIOE의 Base Address인 '0x40011400' 번지에 각 레지스터의 Address offset 값을 더한 주소로 직접 값을 입력하여 제어하고 있다. 주석 문을 읽지 않고 위의 코드만으로는 어떤 기능을 하는 구문인지 알아채기가 쉽지 않다. 또한 사용자가 사용하려는 레지스터의 Base Address와 Address Offset을 모두 알고 있어야 하므로 여간 번거로운 일이 아니다.

> *** 참고사항**
>
> - 'volatile' 이란 '비휘발성의' 라는 뜻으로 언제든 값이 변할 수 있다는 것을 의미한다. 컴파일러가 컴파일을 하는 동안 Optimization 설정 값에 따라 최적화 과정을 거치게 되는데, 컴파일러가 판단하기에 필요 없다고 판단되는 코드는 무시하여 코드의 크기 등을 효율적으로 만들게 된다. 따라서 최적화 과정에서 사용자가 꼭 필요한 코드를 컴파일러가 무시하지 않도록 하기 위해, 이 값은 언제든 변할 수 있으므로 절대 효율적이라 판단하여 무시하지 말고 적혀있는 그대로 컴파일 하라는 뜻으로 'volatile' 이라는 keyword를 붙이는 것이다.

다음은 stm32f10x.h 파일에 정의되어 있는 내용 중 일부이다.

```
#define PERIPH_BASE          ((uint32_t)0x40000000)
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x10000)
...
#define GPIOA_BASE           (APB2PERIPH_BASE + 0x0800)
#define GPIOB_BASE           (APB2PERIPH_BASE + 0x0C00)
#define GPIOC_BASE           (APB2PERIPH_BASE + 0x1000)
#define GPIOD_BASE           (APB2PERIPH_BASE + 0x1400)
#define GPIOE_BASE           (APB2PERIPH_BASE + 0x1800)
...
#define GPIOA             ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB             ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC             ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD             ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE             ((GPIO_TypeDef *) GPIOE_BASE)
...
typedef struct
```

```
{
  __IO uint32_t CRL
  __IO uint32_t CRH
  __IO uint32_t IDR
  __IO uint32_t ODR
  __IO uint32_t BSRR
  __IO uint32_t BRR
  __IO uint32_t LCKR
} GPIO_TypeDef
```

구조체에 정의된 레지스터의 순서는 메모리 주소의 순서와 동일하다. 즉, CRL의 Address offset은 0x0000이었으며, CRH의 offset은 0x0004, BSRR은 0x0010, BRR은 0x0014이다. 구조체의 각 멤버를 4Byte(32bit)로 정의하고, GPIOx_BASE 주소 값을 GPIO_TypeDef 구조체 포인터로 Type casting 하여 구조체의 각 멤버(32bit)에 접근하는 것과 실제 메모리의 각 레지스터(32bit)에 접근하는 것이 동일하도록 되어 있다.

앞으로의 실습 예제에서는 레지스터의 주소를 이용하여 직접 접근하지 않고, 위에서 설명한 구조체의 멤버를 이용하여 각 레지스터에 접근하는 방식을 사용할 것이다. 또한 STM32는 레지스터 설정과 Bit 연산에 관련된 내용을 라이브러리 함수로 제공하여 개발자에게 코드 작성의 편의를 제공하고 있으므로 이를 이용하여 실습을 진행한다. 모든 라이브러리 함수를 여기서 설명할 수는 없으므로 실습을 위해 필요한 라이브러리 함수 몇 가지에 대해서만 살펴보도록 한다.

1. GPIO_Init

GPIO Port를 초기화하고 모드를 설정한다.

void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)	
▪ 기능	GPIO Port Initialize
▪ 인수	GPIOx - 제어할 GPIOx의 (GPIO_TypeDef*)Base Address GPIO_InitStruct - 초기화할 정보를 담고 있는 구조체
▪ stm32f10x_gpio.h 참고	

GPIO_InitTypeDef 구조체는 GPIO_Init 함수 호출 시 설정 값을 한꺼번에 넘겨주기 위한 구조체이다. GPIO_InitTypeDef 구조체의 원형은 다음과 같다.

```
typedef struct
{
  uint16_t GPIO_Pin;
  GPIOSpeed_TypeDef GPIO_Speed;
  GPIOMode_TypeDef GPIO_Mode;
}GPIO_InitTypeDef;
```

제어 할 Pin 번호와 Mode, Output일 경우의 주파수 설정을 할 수 있다. 구조체에 값을 넣어 Init 함수를 호출하면 구조체의 값에 따라 GPIOx_CRL, GPIOx_CRH 레지스터가 설정된다.

2. GPIO_SetBits

함수 명 그대로 해당 Bit를 Set 시킨다. 한 개 이상의 Pin을 한꺼번에 Set 할 수 있다.

void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)	
▪ 기능	GPIO Bits Set
▪ 인수	GPIOx - 제어할 GPIOx의 (GPIO_TypeDef*)Base Address GPIO_Pin - Set 시킬 Pin의 번호
▪ stm32f10x_gpio.h 참고	

3. GPIO_ResetBits

해당 Bit를 Reset 시킨다. 한 개 이상의 Pin을 한꺼번에 Reset 할 수 있다.

void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)	
▪ 기능	GPIO Bits Reset
▪ 인수	GPIOx - 제어할 GPIOx의 (GPIO_TypeDef*)Base Address GPIO_Pin - Reset 시킬 Pin의 번호
▪ stm32f10x_gpio.h 참고	

4. GPIO_ReadInputData

IDR 레지스터의 값을 한꺼번에 읽어와 반환한다.

uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx)	
▪ 기능	Read IDR (Input Data Register)
▪ 인수	GPIOx - 제어할 GPIOx의 (GPIO_TypeDef*)Base Address
▪ 반환 값	IDR의 15:0번 비트에 저장된 16bit unsigned int 값
▪ stm32f10x_gpio.h 참고	

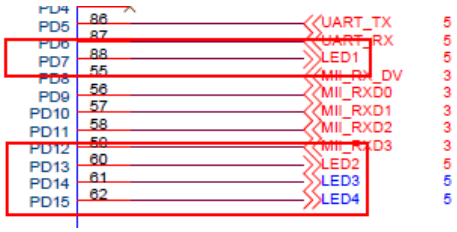
5. GPIO_ReadInputDataBits

IDR 레지스터의 해당 bit 값을 읽어와 반환한다.

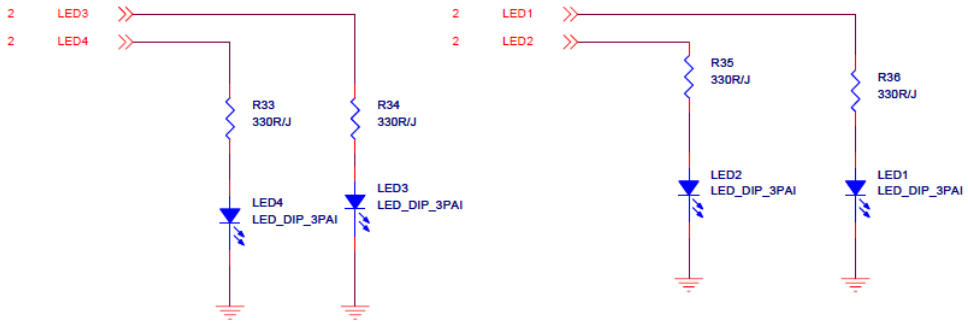
uint8_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)	
▪ 기능	Read IDR (Input Data Register)
▪ 인수	GPIOx - 제어할 GPIOx의 (GPIO_TypeDef*)Base Address GPIO_Pin - 읽어 올 Pin의 번호
▪ 반환 값	IDR의 GPIO_Pin의 값, 0 또는 1
▪ stm32f10x_gpio.h 참고	

2.4 LED 제어 실습하기

지금까지 배운 GPIO에 대한 내용으로 실습 보드의 LED를 제어하는 코드를 작성해보자. LED를 제어하기 위해서 우선 실습 보드의 회로도를 살펴봐야 한다.



회로도에서 알 수 있듯이 실습 보드에 연결 된 LED 4개는 PortD의 7번, 13번, 14번, 15번 Pin에 연결되어 있다. LED의 경우 전원에 연결되어 있는지, GND에 연결되어 있는지에 따라 LED를 켜기 위한 값이 달라지는데, 실습 보드에서 LED는 GND에 연결되어 있으므로 Pin에 '1' 값을 내보내야 LED를 켤 수 있다. 다음 그림은 LED의 회로도를 나타낸 것이다.



실습 1. LED 제어 실습하기

1초마다 LED가 켜졌다 꺼지도록 소스 코드를 작성하라.

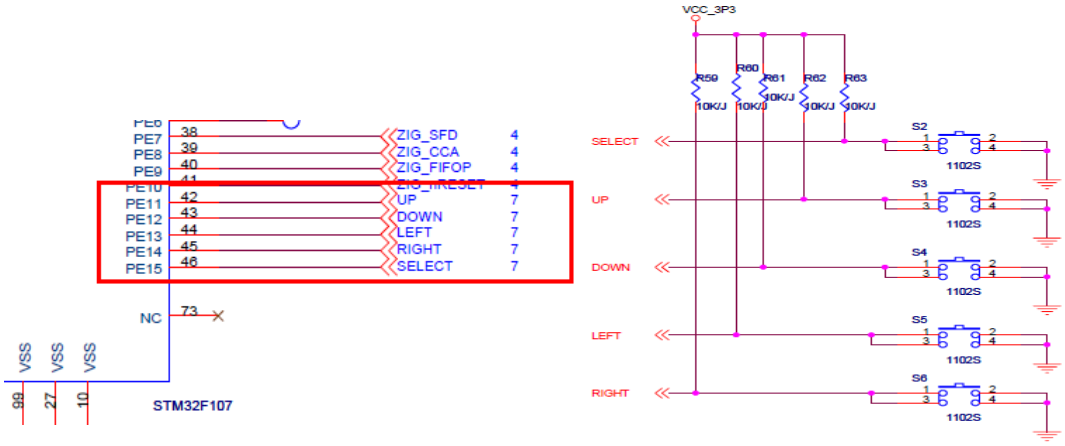
▣ Source Code (Project Name - GPIO_LED_Ctrl) - main.c

```
#include "stm32f10x.h"

void Led_On(void);
void Led_Off(void);

int main(void)
{
        int x = 0;
        GPIO_InitTypeDef GPIO_InitStructure;     /* InitTypeDef 구조체 선언 */

        /* PortD에 Clock을 공급하여 활성화 */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
```

```
        /* InitTypeDef 구조체 멤버에 값 설정 */
        /* 제어 할 Pin 번호 설정 */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
        /* Push/Pull Output Mode 설정 */
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        /* 통신 속도 설정 */
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        /* GPIO_Init 함수 호출, Port 번호와 InitTypeDef 구조체 포인터를 매개변수로 전달 */
        GPIO_Init(GPIOD, &GPIO_InitStructure);

        while(1)
        {
                Led_On();
                for(x = 0; x < 0xFFFFF; x++);          /* 1초 Delay를 위한  Dummy Loop */
                Led_Off();
                for(x = 0; x < 0xFFFFF; x++);          /* 1초 Delay를 위한  Dummy Loop */
        }
        return 0;
}

/* ============ LED On 함수 구현 ============ */
void Led_On(void)
{
        /* GPIOD_BSRR의 해당 bit를 set하여 GPIOD_ODR을 Set */
        GPIO_SetBits(GPIOD, GPIO_Pin_7);
        GPIO_SetBits(GPIOD, GPIO_Pin_13);
        GPIO_SetBits(GPIOD, GPIO_Pin_14);
        GPIO_SetBits(GPIOD, GPIO_Pin_15);
}

/* ============ LED Off 함수 구현 =========== */
void Led_Off(void)
{
        /* GPIOD_BRR의 해당 bit를 set하여 GPIOD_ODR을 Clear */
        GPIO_ResetBits(GPIOD, GPIO_Pin_7);
        GPIO_ResetBits(GPIOD, GPIO_Pin_13);
        GPIO_ResetBits(GPIOD, GPIO_Pin_14);
        GPIO_ResetBits(GPIOD, GPIO_Pin_15);
}
```

2.5 Key 제어 실습하기

Key 입력을 받아 LED를 제어하는 소스를 작성해보자. 다음은 실습 보드의 Chip에 Key가 연결된 모습과 Key의 회로도를 나타낸 것이다.



Key는 실습 보드의 PortE 11번 Pin부터 15번 Pin까지 연결되어 있다. Key의 회로도를 살펴보면 스위치와 Pin 사이에 Vcc가 연결되어 있다. 따라서 Key가 눌려지지 않은 상태에서는 '1' 이 입력되고, Key가 눌려지면 '0' 이 입력될 것이다.

실습 2. Key 제어 실습하기

Up, Down, Left, Right Key가 입력되면 순서대로 LED를 켜고 Select Key가 입력되면 LED가 모두 꺼지도록 소스 코드를 작성하라.

▣ Source Code (Project Name - GPIO_Key_Ctrl) - main.c

```
#include "stm32f10x.h"

/* =========== LED 제어를 위한 정의 ========== */
#define   GPIO_LED        GPIOD
#define   RCC_LED         RCC_APB2Periph_GPIOD
#define   LED1            GPIO_Pin_7
#define   LED2            GPIO_Pin_13
#define   LED3            GPIO_Pin_14
#define   LED4            GPIO_Pin_15
/* ===================================== */

/* ========== Key 제어를 위한 정의 ============ */
#define   GPIO_KEY        GPIOE
#define   RCC_KEY         RCC_APB2Periph_GPIOE
#define   KEY_UP          GPIO_Pin_11
#define   KEY_DOWN        GPIO_Pin_12
```

```
#define   KEY_LEFT          GPIO_Pin_13
#define   KEY_RIGHT         GPIO_Pin_14
#define   KEY_SELECT        GPIO_Pin_15
/* ===================================== */

int main(void)
{
        /* Port 설정을 위한 구조체 선언 */
        GPIO_InitTypeDef GPIO_InitStruct;

        /* LED와 Key Port 활성화를 위해 Port에 Clock 공급 */
        RCC_APB2PeriphClockCmd(RCC_LED | RCC_KEY, ENABLE);

        GPIO_InitStruct.GPIO_Pin = LED1 | LED2 | LED3 | LED4;    /* Set Led1~4 Pin */
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;            /* Push/Pull Output mode */
        GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;           /* 50MHz 통신 속도 */
        GPIO_Init(GPIO_LED, &GPIO_InitStruct);

        GPIO_InitStruct.GPIO_Pin = KEY_UP | KEY_DOWN | KEY_LEFT | KEY_RIGHT
                                 | KEY_SELECT;                   /* Set Key Pin */
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;       /* Floating Input mode */
        GPIO_Init(GPIO_KEY, &GPIO_InitStruct);

        /* 초기 상태 설정, LED All Off */
        GPIO_ResetBits(GPIO_LED, LED1);
        GPIO_ResetBits(GPIO_LED, LED2);
        GPIO_ResetBits(GPIO_LED, LED3);
        GPIO_ResetBits(GPIO_LED, LED4);

        while(1)
        {
                /* Key 입력을 체크하여 Led On/ Off */
                /* Push Up Key -> LED1 On */
                if(!GPIO_ReadInputDataBit(GPIO_KEY, KEY_UP))
                        GPIO_SetBits(GPIO_LED, LED1);
                /* Push Down Key -> LED2 On */
                else if(!GPIO_ReadInputDataBit(GPIO_KEY, KEY_DOWN))
                        GPIO_SetBits(GPIO_LED, LED2);
                /* Push Left Key -> LED3 On */
                else if(!GPIO_ReadInputDataBit(GPIO_KEY, KEY_LEFT))
```

```
                        GPIO_SetBits(GPIO_LED, LED3);
                /*Push Right Key -> LED4 On */
                else if(!GPIO_ReadInputDataBit(GPIO_KEY, KEY_RIGHT))
                        GPIO_SetBits(GPIO_LED, LED4);
                /* Push Select Key -> LED All Off */
                else if(!GPIO_ReadInputDataBit(GPIO_KEY, KEY_SELECT))
                {
                        GPIO_ResetBits(GPIO_LED, LED1);
                        GPIO_ResetBits(GPIO_LED, LED2);
                        GPIO_ResetBits(GPIO_LED, LED3);
                        GPIO_ResetBits(GPIO_LED, LED4);
                }
        }
        return 0;
}
```

실습 3. GPIO Interrupt 제어하기

3.1 GPIO Interrupt 개요

STM32 Chip의 Core인 Cortex-M3에는 많은 System Exception과 외부 IRQ를 가지고 있다. ARM7과 비교해서 FIQ를 가지고 있지는 않지만, Interrupt 우선순위에 대한 처리와 Nested Vectored Interrupt를 이용해 쉽게 FIQ와 비슷하게 동작할 수 있도록 구현 가능하다.

Cortex-M3는 내부 Exception Source를 포함해서 모두 256개의 Interrupt Source를 가진다. 이것은 Chip 제조 시 미리 정의되며, 제조사에서는 최대 240개의 외부 Interrupt Source를 사용할 수 있다.

NVIC는 IRQ handler가 수행될 때 Vector를 Index되어 있는 곳으로 바로 이동시키기 때문에 인터럽트 지연시간을 획기적으로 줄여줄 뿐 아니라 어떤 Handler가 수행되어야 하는지를 결정하는 소프트웨어도 필요 없게 만든다. Nested와 관련된 처리 역시 하드웨어적으로 이루어지기 때문에 별도의 소프트웨어 처리가 필요 없다. 이것은 처리 속도를 굉장히 빠르게 하여 STM32의 큰 장점이 된다.

Interrupt 발생 시 CPU는 자동으로 Processor의 상태를 Exception의 Stack에 저장하고, 빠져 나올 때 Interrupt Service Routine(ISR)의 마지막에서 복구시킨다. 이와 같이 소프트웨어적으로 처리해줘야 했던 부분을 하드웨어에서 자동으로 처리해 줌으로써 일반적인 C 함수를 Interrupt Handler로 사용할 수 있다는 이점을 가져다주는 것과 동시에 IRQ 지연 시간도 훨씬 줄어들도록 한다.

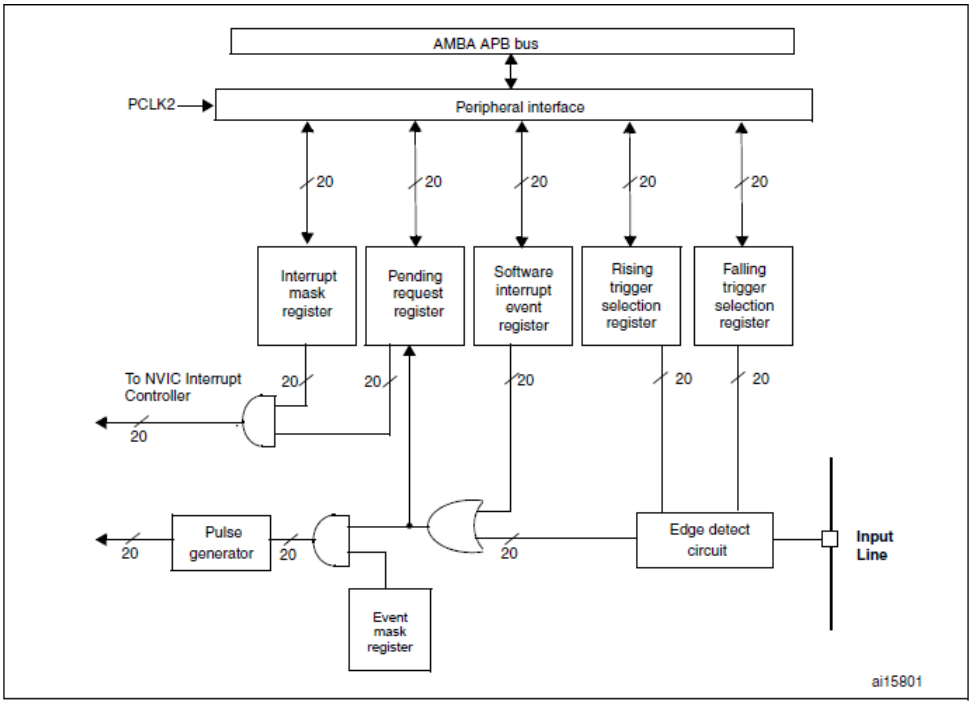
또한 Cortex-M3는 Tail-Chaining이라는 기능을 제공한다. 이것은 연속적으로 발생하는 Interrupt에 대해 Interrupt 발생 시점마다 매번 상태를 Stack에 저장하고 복구하는 일을 수행하는 것이 아니라, 이전 값을 그대로 유지하여 다음 Interrupt의 처리 속도를 빠르게 만드는 기능이다. 이것은 Interrupt가 연속적으로 일어나는 것이 확실할 때, 이전 Interrupt에서 복구되는 값이 다음 Interrupt에서 저장되는 값과 동일한 것이라고 판단하기 때문이다.

다음은 STM32f10x의 데이터 시트에 나와 있는 Interrupt Vector Table 중 External Interrupt에 관련된 내용만을 나타낸 것이다.

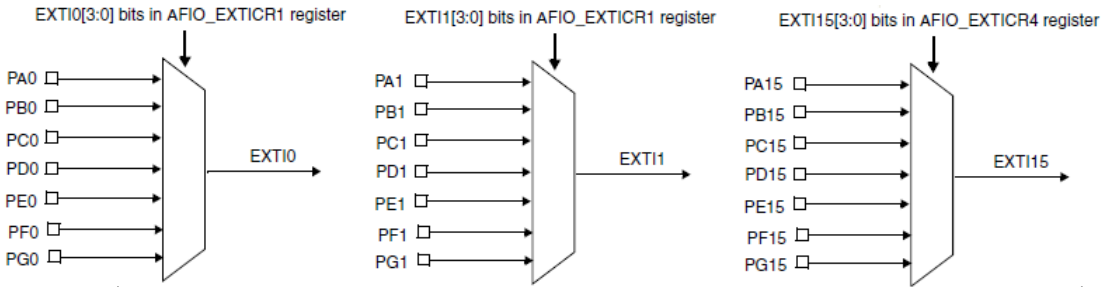
Position	Priority	Type of priority	Acronym	Description	Address
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0

External Interrupt/Event Controller는 20개의 Edge Detectors로 구성되어 있으며 Event나 Interrupt를 구동시키는 부분이다. 각 라인은 독립적으로 구성이 가능하고 특정한 Trigger Event(Rising 또는 Falling Edge, 혹은 양 쪽 모두)에 대해서도 처리가 가능하다. 각 라인에 대해서 역시 독립적으로 Masking도 가능하다.

다음은 External Interrupt/Event Controller의 Block Diagram을 나타낸 것이다.



20개의 Detectors 중 0~15번 Detector는 GPIOx의 각 핀에 연결되어 있다. Portx의 n번 핀은 n번 Edge Detector에 연결된다. 16번 Detector는 PVD 핀, 17번 Detector는 RTC Alarm Event, 18번은 USB Wakeup Event, 19번 Detector는 Ethernet Wakeup Event에 연결되어 있다. 다음 그림은 External Interrupt/Event GPIO Mapping을 나타낸 것이다.



그림에서 볼 수 있듯이 AFIO_EXTICRn 레지스터에서 Port에 대한 설정을 할 수 있다. AFIO_EXTICRn 레지스터에 대한 내용은 다음 절에서 자세히 살펴보도록 한다.

3.2 GPIO Interrupt 레지스터

GPIO Interrupt를 제어하기 위해 사용되는 레지스터에 대해서 알아보자.

1. AFIO_EXTICRn (External Interrupt Configuration Register n) (n : 1~4)

Base Address : AFIO, Address offset : 0x08~0x14, Reset Value : 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

4bit가 각각 어떤 GPIO와 연관 시킬지를 설정하는 레지스터이다. AFIO_EXTICR1은 EXTI0~3, AFIO_EXTICR2는 EXTI4~7, AFIO_EXTICR3은 EXTI8~11, AFIO_EXTICR4는 EXTI12~15를 설정한다. 4bit로 설정되는 값은 다음과 같다.

Bits 15:0 **EXTIx[3:0]:** EXTI x configuration
These bits are written by software to select the source input for EXTIx external interrupt.
0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin

2. EXTI_IMR (External Interrupt Mask Register)

Address offset : 0x00, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

각 Interrupt Request를 Making 한다. '0' 으로 설정될 경우 해당 EXTI Line이 Mask되어 Interrupt가 발생하지 않는다. 초기 값이 모두 0이므로, Interrupt가 발생하도록 하기 위해서는 해당 bit를 '1' 로 설정해줘야 한다.

3. EXTI_RTSR (External Interrupt Rising Trigger Selection Register)

Address offset : 0x08, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

각 Input Line의 Rising Trigger를 Enable한다. EXTI_FTSR의 bit와 동시에 설정될 수 있다.

4. EXTI_FTSR (External Interrupt Falling Trigger Selection Register)

Address offset : 0x0C, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

각 Input Line의 Falling Trigger를 Enable한다. EXTI_RTSR의 bit와 동시에 설정될 수 있다.

5. EXTI_PR (External Interrupt Pending Register)

Address offset : 0x14, Reset Value : undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												PR19	PR18	PR17	PR16
												rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

이 레지스터의 특정 bit가 '1' 이라는 것은 선택된 Trigger Request가 발생했다는 것을 의미한다. 우리가 설정했던 Rising 또는 Falling Trigger가 발생했다는 것이다. 하드웨어에 의해 자동으로 Clear되지 않으므로 인터럽트를 확인하면 '1' 을 쓰는 것으로 해당 bit를 Clear 해줘야 한다.

3.3 GPIO Interrupt 라이브러리 함수

Interrupt를 제어하기 위해서는 NVIC(Nested Vector Interrupt Controller) 설정과 EXTI(External Interrupt) 설정을 해야 한다.

1. NVIC_SetVectorTable

Vector Table offset Register(SCB_VTOR)를 설정하는 함수이다. 보통 Base Address는 RAM 영역(0x2000 0000)이거나 FLASH 영역(0x0800 0000)이다.

void NVIC_SetVectorTable(uint32_t NVIC_VectTab, uint32_t offset)	
▪ 기능	Set Vector table offset register
▪ 인수	NVIC_VectTab - Vector Table의 Base Address offset - offset
▪ misc.c 참고	

NVIC_VectTab의 값은 다음과 같이 정의되어 있다.

```
#define NVIC_VectTab_RAM          ((uint32_t)0x20000000)
#define NVIC_VectTab_FLASH        ((uint32_t)0x08000000)
```

2. NVIC_PriorityGroupConfig

그룹 우선순위를 설정한다.

void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)	
▪ 기능	Set Priority Group
▪ 인수	NVIC_PriorityGroup - 그룹 우선순위 값
▪ misc.c 참고	

3. NVIC_Init

void NVIC_Init(NVIC_InitTypeDef* NVIC_IntiStruct)	
▪ 기능	NVIC Initialization
▪ 인수	NVIC_InitStruct - 초기화 설정 정보를 담고 있는 구조체 포인터
▪ misc.c 참고	

4. GPIO_EXTILineConfig

앞에서 설명한 AFIO_EXTICRn 레지스터를 설정하여 인터럽트 Port와 Pin을 설정한다. 단, Pin을 중첩하여 설정할 수 없다.

void GPIO_EXTILineConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource)	
▪ 기능	Set AFIO_EXTICRn
▪ 인수	GPIO_PortSource - 인터럽트를 활성화 할 Port 설정 GPIO_PinSource - 인터럽트를 활성화 할 Pin 번호 설정
▪ stm32f10x_gpio.c 참고	

5. EXTI_Init

void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct)	
▪ 기능	EXTI Initialization
▪ 인수	EXTI_InitStruct - 초기 설정 값을 담고 있는 구조체 포인터
▪ stm32f10x_exti.c 참고	

EXTI_InitTypeDef 구조체의 원형은 다음과 같다.

```
typedef struct
{
  uint32_t EXTI_Line
  EXTIMode_TypeDef EXTI_Mode
  EXTITrigger_TypeDef EXTI_Trigger
  FunctionalState EXTI_LineCmd
}EXTI_InitTypeDef
```

6. EXTI_GetITStatus

EXTI_PR 레지스터를 통해 해당 Line에 인터럽트가 발생했는지 확인 후 결과를 반환한다.

ITStatus EXTI_GetITStatus(uint32_t EXTI_Line)	
▪ 기능	해당 Line에 인터럽트 여부를 확인
▪ 인수	EXTI_Line - 인터럽트를 확인 할 Line의 번호
▪ 반환 값	ITStatus - 인터럽트의 발생 여부 반환 (SET/RESET)
▪ stm32f10x_exti.c 참고	

7. EXTI_ClearITPendingBit

void EXTI_ClearITPendingBit(uint32_t EXTI_Line)	
▪ 기능	EXTI_PR 레지스터의 해당 Line bit를 Clear 한다.
▪ 인수	EXTI_Line - Clear 할 Line의 번호
▪ stm32f10x_exti.c 참고	

8. EXTI1_IRQHandler ~ EXTI4_IRQHandler, EXTI5_9_IRQHandler, EXTI15_10_IRQHandler

STM32 Chip의 경우 ISR 함수에 대한 문법이 따로 존재하지 않는다. 단지 IRQHandler 함수에 내용을 구현해주면 될 뿐이다. 이유는 앞 절에서 설명한 대로 Interrupt에 대한 처리를 하드웨어에서 자동으로 해주기 때문이다.

ESTIn_IRQHandler 함수는 EXTIn번 인터럽트가 발생했을 경우 처리할 내용이 구현되는 곳이다. 5~15번 인터럽트의 경우 함수 내에서 EXTI_GetITStatus() 함수를 통해 인터럽트가 발생한 Pin을 확인 후 따로 처리할 수 있다. 함수의 위치는 'System\ src\ stm32f10x_it.c' 이다. stm32f10x_it.c 파일에서는 EXTI 외에도 다른 인터럽트 처리 함수를 구현할 수 있다.

void EXTIn_IRQHandler(void)	
▪ 기능	EXTIn 번 인터럽트 처리 함수
▪ 인수	없음
▪ stm32f10x_it.c 참고	

3.4 GPIO Interrupt 제어 실습하기

2장에서 우리는 Key의 입력을 받아 LED를 켜고 끄는 실습을 했었다. 앞에서는 while문을 이용하여 Key의 입력을 체크하는 방법이었지만, 이번에는 Key 입력 Pin의 인터럽트를 이용하여 LED를 제어하는 실습을 해보자.

실습 1. GPIO Interrupt 제어 실습하기

인터럽트를 이용하여 Key 입력을 감지하고 해당 LED를 켜고 끄는 소스 코드를 작성하라.

* Header 파일 추가

▣ Source Code (Project Name - Interrupt_Ctrl) - main.h

```
#include "stm32f10x_exti.h"

/* =========== LED 제어를 위한 정의 ========== */
#define   GPIO_LED              GPIOD
#define   RCC_LED               RCC_APB2Periph_GPIOD
#define   LED1                  GPIO_Pin_7
#define   LED2                  GPIO_Pin_13
#define   LED3                  GPIO_Pin_14
#define   LED4                  GPIO_Pin_15
#define   LED_ALL               (LED1 | LED2 | LED3 | LED4)
/* ===================================== */

/* ========== Key 제어를 위한 정의 =========== */
#define   GPIO_KEY              GPIOE
#define   RCC_KEY               RCC_APB2Periph_GPIOE
#define   KEY_UP                GPIO_Pin_11
#define   KEY_DOWN              GPIO_Pin_12
#define   KEY_LEFT              GPIO_Pin_13
#define   KEY_RIGHT             GPIO_Pin_14
#define   KEY_SELECT            GPIO_Pin_15
#define   KEY_PORTSOURCE        GPIO_PortSourceGPIOE
#define   KEY_PINSOURCE         GPIO_PinSource11
/* ===================================== */

/* ==== External Interrupt 제어를 위한 정의 ==== */
#define   RCC_AFIO              RCC_APB2Periph_AFIO

#define   EXTI_UKEY_LINE        EXTI_Line11
#define   EXTI_DKEY_LINE        EXTI_Line12
#define   EXTI_LKEY_LINE        EXTI_Line13
```

```
#define   EXTI_RKEY_LINE          EXTI_Line14
#define   EXTI_SKEY_LINE          EXTI_Line15
/* ===================================== */
```

* 메인 함수 구현

▣ Source Code (Project Name - Interrupt_Ctrl) - main.c

```
#include <stm32f10x.h>
#include <main.h>

/* ============== 함수 선언 ================ */
void GPIO_Configuration();
void NVIC_Configuration();
void EXTI_Configuration();
/* ===================================== */

/* ============ Main 함수 구현 ============== */
int main(void)
{
        GPIO_Configuration();
        NVIC_Configuration();
        EXTI_Configuration();

        GPIO_ResetBits(GPIO_LED, LED_ALL);          /* LED All Off */

        while(1)
        {
                ;                                   /* Loop for Wait Interrupt */
        }
        return 0;
}

/* ========== GPIO 설정 함수 구현 ============ */
void GPIO_Configuration()
{
        GPIO_InitTypeDef GPIO_InitStruct;

        /* LED와 Key Port 활성화를 위해 Port에 Clock 공급 */
        RCC_APB2PeriphClockCmd(RCC_LED | RCC_KEY, ENABLE);

        /* LED Configuration */
```

```
        GPIO_InitStruct.GPIO_Pin = LED1 | LED2 | LED3 | LED4;    /* Set Led1~4 Pin */
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;            /* Push/Pull Output mode */
        GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;           /* 50MHz 통신 속도 */
        GPIO_Init(GPIO_LED, &GPIO_InitStruct);

        /* Key Configuration */
        GPIO_InitStruct.GPIO_Pin = KEY_UP | KEY_DOWN | KEY_LEFT | KEY_RIGHT
                                 | KEY_SELECT;                   /* Set Key Pin */
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;       /* Floating Input mode */
        GPIO_Init(GPIO_KEY, &GPIO_InitStruct);
}

/* ========== NVIC 설정 함수 구현 ============ */
void NVIC_Configuration()
{
        NVIC_InitTypeDef NVIC_InitStruct;

        NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

        /* NVIC Configuration */
        NVIC_InitStruct.NVIC_IRQChannel = EXTI15_10_IRQn;
        NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStruct);
}

/* ========== EXTI 설정 함수 구현 ============ */
void EXTI_Configuration()
{
        int i;
        EXTI_InitTypeDef EXTI_InitStruct;

        RCC_APB2PeriphClockCmd(RCC_AFIO, ENABLE);

        for(i=0; i<5; i++)    /* PIN 11~15 Interrupt Set*/
        {
                GPIO_EXTILineConfig(KEY_PORTSOURCE, KEY_PINSOURCE+i);
        }
```

```
        EXTI_InitStruct.EXTI_Line = EXTI_UKEY_LINE | EXTI_DKEY_LINE | EXTI_LKEY_LINE
                                    | EXTI_RKEY_LINE | EXTI_SKEY_LINE;
        EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
        EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;        /* Falling Edge */
        EXTI_InitStruct.EXTI_LineCmd = ENABLE;
        EXTI_Init(&EXTI_InitStruct);
}
```

* 인터럽트 처리 함수 구현

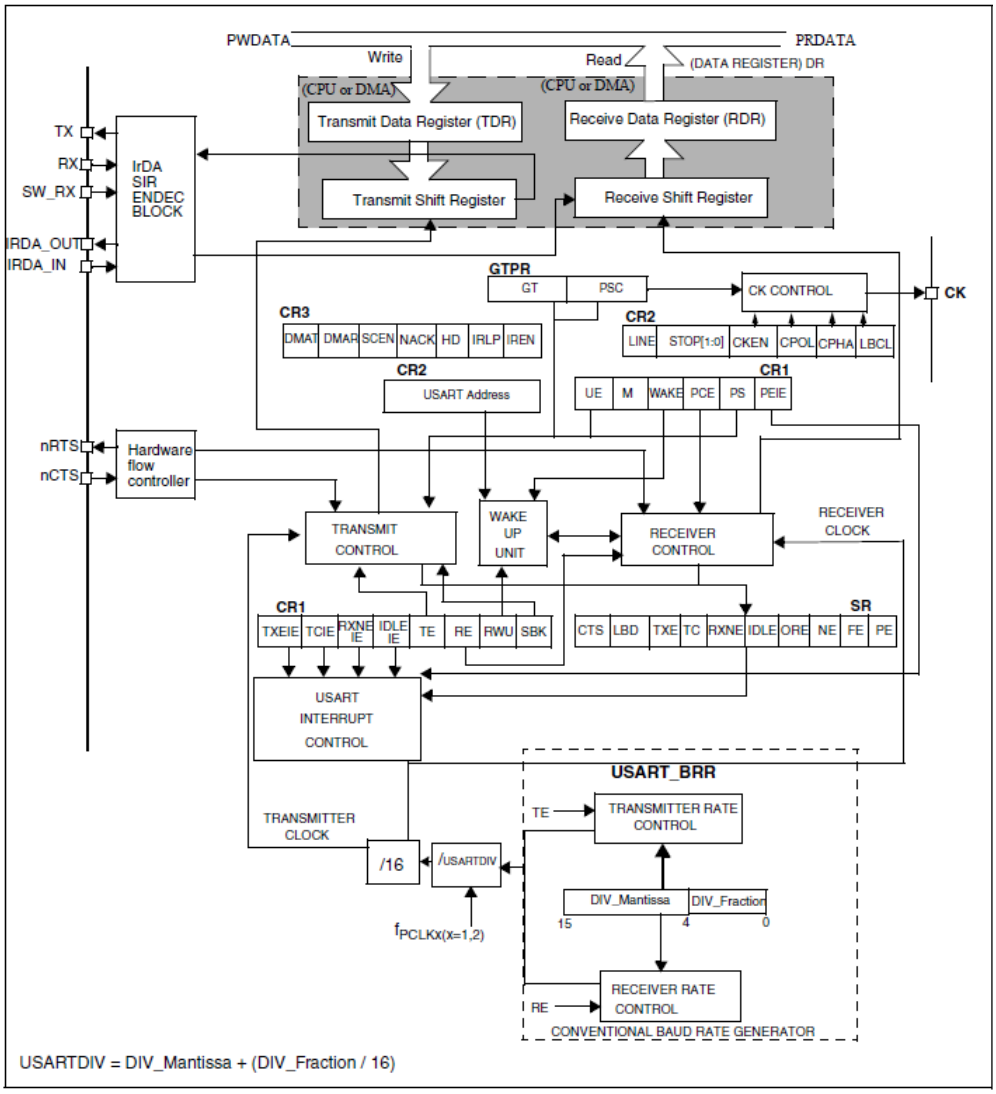
▣ Source Code (Project Name - Interrupt_Ctrl) - System/src/stm32f10x_it.c

```
void EXTI15_10_IRQHandler(void)
{
        /* Push Up Key, LED1 On */
        if(EXTI_GetITStatus(EXTI_UKEY_LINE) != RESET) {
                EXTI_ClearITPendingBit(EXTI_UKEY_LINE);
                GPIO_SetBits(GPIO_LED, LED1);
        }
        /* Push Down Key, LED2 On */
        else if(EXTI_GetITStatus(EXTI_DKEY_LINE) != RESET) {
                EXTI_ClearITPendingBit(EXTI_DKEY_LINE);
                GPIO_SetBits(GPIO_LED, LED2);
        }
        /* Push Left Key, LED3 On */
        else if(EXTI_GetITStatus(EXTI_LKEY_LINE) != RESET) {
                EXTI_ClearITPendingBit(EXTI_LKEY_LINE);
                GPIO_SetBits(GPIO_LED, LED3);
        }
        /* Push Right Key, LED4 On */
        else if(EXTI_GetITStatus(EXTI_RKEY_LINE) != RESET) {
                EXTI_ClearITPendingBit(EXTI_RKEY_LINE);
                GPIO_SetBits(GPIO_LED, LED4);
        }
        /* Push Select Key, LED All Off */
        else if(EXTI_GetITStatus(EXTI_SKEY_LINE) != RESET) {
                EXTI_ClearITPendingBit(EXTI_SKEY_LINE);
                GPIO_ResetBits(GPIO_LED, LED_ALL);
        }
}
```

실습 4. USART 통신 제어하기

4.1 USART 통신 개요

STM32 Chip은 USART(Universal Synchronous/Asynchronous Receiver Transmitter) 통신을 지원한다. USART 통신은 2개의 Pin으로 다른 장치와 통신하는 방식을 말한다. 다음은 USART의 Block diagram을 나타낸 것이다.



Start bit와 Stop bit를 가지며, Data는 8bit 또는 9bit로 구성된다. 송수신 데이터에 대한 Parity Controller가 가능하고, Overrun, Noise, Frame, Parity에 대한 Error Check flag가 존재한다. 각 레지스터에 대한 내용은 다음 절에서 자세히 설명하도록 하겠다.

4.2 USART 레지스터

USART에 대한 레지스터 주소 영역은 다음과 같다. (USART와 UART가 있는데, UART는 Synchronous 통신을 지원하지 않는 채널을 뜻한다.)

Boundary address	Peripheral
0x4001 3800 - 0x4001 3BFF	USART1
0x4000 5000 - 0x4000 53FF	UART5
0x4000 4C00 - 0x4000 4FFF	UART4
0x4000 4800 - 0x4000 4BFF	USART3
0x4000 4400 - 0x4000 47FF	USART2

1. USART_SR (USART Status Register)

Address offset : 0x00, Reset Value : 0x0000 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
Res.						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

레지스터의 각 bit는 USART 통신 상태 정보를 나타내고 있는데, 여기서는 실습에 필요한 부분만 살펴보도록 하겠다.

- **TXE** (Transmit Data Register Empty) : TDR 레지스터의 내용이 Shift Register로 전달이 완료될 때 하드웨어에 의해 자동으로 '1' 이 Set 된다. 또한 USART_DR 레지스터에 데이터를 넣으면 '0' 으로 Clear 된다.
  0 : Data is not transferred to the shift register
  1 : Data is transferred to the shift register
- **RXNE** (Receiver Data Register Not Empty) : RDR 레지스터에 읽지 않은 내용이 있을 때 '1' 로 설정된다. 이 값은 USART_DR 레지스터의 데이터를 읽어오면 하드웨어에 의해 '0' 으로 Clear 된다.
  0 : Data is not received
  1 : Received data is ready to be read

2. USART_DR (USART Data Register)

Address offset : 0x04, Reset Value : Undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							DR[8:0]								
Res.							rw	rw	rw	rw	rw	rw	rw	rw	rw

USART의 데이터는 8bit 또는 9bit이므로 9bit만이 유효한 값을 가진다. 이전 절의 Block Diagram에서 볼 수 있듯이 송수신시 모두 USART_DR을 이용하지만 내부적으로 TDR과 RDR로 나뉘어져 있기 때문에, 송수신이 독립적으로 동작할 수 있다.

3. USART_BRR (USART Baud Rate Register)

Address offset : 0x08, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

통신의 Baud Rate에 관련된 값을 설정한다. 자세한 내용은 해당 라이브러리 함수를 참조하도록 한다.

4. USART_CR1 (USART Control Register 1)

Address offset : 0x0C, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
Res.		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

USART 설정에 관련된 레지스터이다. 실습에 필요한 부분만 살펴보도록 한다.

- **UE** (USART Enable) : USART를 활성화 시킨다.
  0 : USART Prescaler and output disable
  1 : USART Enable
- **M** (Word Length bit) : Data의 길이를 지정한다.
  0 : 1 Start bit, 8 Data bit, n Stop bit
  1 : 1 Start bit, 9 Data bit, n Stop bit
- **PCE** (Parity Control Enable) : Parity Control을 설정한다. 이 bit가 활성화 되면 데이터의 MSB 위치에 Parity bit가 위치하게 된다.
  0 : Parity control disabled
  1 : Parity control enable
- **PS** (Parity Selection) : PCE가 '1' 일 경우 Parity의 종류를 선택한다.
  0 : Even Parity
  1 : Odd Parity
- **TE** (Transmitter Enable) : USART의 Transmitter를 활성화 시킨다.
  0 : Transmitter is disabled
  1 : Transmitter is enabled
- **RE** (Receiver Enable) : USART의 Receiver를 활성화 시킨다.
  0 : Receiver is disabled
  1 : Receiver is enabled

5. USART_CR2 (USART Control Register 2)

Address offset : 0x10, Reset Value : 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

- **STOP[1:0]** (Stop bit) : Stop bit에 대한 설정은 다음과 같다.

  00 : 1 Stop bit

  01 : 0.5 Stop bit

  10 : 2 Stop bit

  11 : 1.5 Stop bit

4.3 USART 라이브러리 함수

USART 제어를 위해 필요한 라이브러리 함수에 대해서 알아보자.

1. USART_Init

USART 설정을 초기화한다.

void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)	
▪ 기능	USART의 해당 채널을 설정 값을 초기화한다.
▪ 인수	USARTx - 설정할 USART 채널 USART_InitStruct - 설정 값을 담고 있는 초기화 구조체 포인터
▪ stm32f10x_usart.c 참고	

USART_InitTypeDef 구조체의 원형은 다음과 같다.

```
typedef struct
{
  uint32_t USART_BaudRate
  uint16_t USART_WordLength
  uint16_t USART_StopBits
  uint16_t USART_Parity
  uint16_t USART_Mode
  uint16_t USART_HardwareFlowControl
} USART_InitTypeDef
```

BaudRate를 제외한 각 멤버에는 실제 값이 아니라 설정 레지스터에 대한 Bit Flag가 들어가야 한다. 예를 들어 USART_WordLength에는 '8' 또는 '9'가 아니라 '(uint16_t)0x0000' 또는 '(uint16_t)0x1000'이 들어가야 한다. stm32f10x_usart.h에 이에 대한 설정 값이 다음과 같이 정의되어 있다.

```
#define USART_WordLength_8b              ((uint16_t)0x0000)
#define USART_WordLength_9b              ((uint16_t)0x1000)
```

설정 값에 대한 자세한 정의는 stm32f10x_usart.h 파일을 참고한다.

2. USART_Cmd

void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState)	
▪ 기능	USART를 활성화 시킨다. (USART_CR1의 UE bit 활성화)
▪ 인수	USARTx - USART 채널 NewState - 활성화 여부 (ENABLE or DISABLE)
▪ stm32f10x_usart.c 참고	

3. USART_SendData

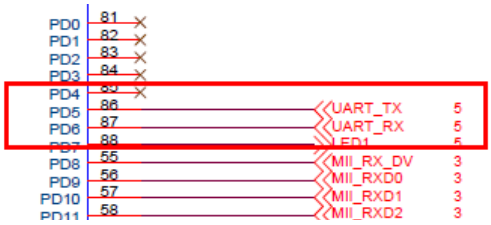
void USART_SendData(USART_TypeDef* USARTx, uint16_t Data)	
▪ 기능	USART의 해당 채널로 데이터를 전송한다.
▪ 인수	USARTx - 데이터를 전송할 USART 채널 Data - 전송할 데이터
▪ stm32f10x_usart.c 참고	

4. USART_GetFlagStatus

FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG)	
▪ 기능	USART의 상태 Flag를 읽어 반환한다.
▪ 인수	USARTx - USART 채널 USART_FLAG - 읽어 올 Bit Flag
▪ 반환 값	FlagStatus - Bit Flag의 값 (SET('1') 또는 RESET('0'))
▪ stm32f10x_usart.c 참고	

4.4 USART 통신 제어 실습하기

USART 통신을 이용해 하이퍼 터미널에 문자열을 출력해보자. 우선 USART 통신을 하기 위해 실습 보드에 USART 연결 상태를 확인해야 한다. 다음은 실습 보드의 회로도 중 USART 부분을 나타낸 것이다.



USART Tx Pin은 PortD의 5번 Pin에, Rx Pin은 PortD의 6번 Pin에 연결되어 있다. STM32f10x의 데이터 시트에서 PD5, 6번 Pin에 대한 내용을 찾아보면 다음과 같이 USART 2번 채널에 Remap 되어 있는 것을 확인할 수 있다.

Alternate functions	USART2_REMAP = 0	USART2_REMAP = 1(1)
USART2_CTS	PA0	PD3
USART2_RTS	PA1	PD4
USART2_TX	PA2	PD5
USART2_RX	PA3	PD6
USART2_CK	PA4	PD7

1. Remap available only for 100-pin and 144-pin packages.

GPIO에 관한 내용에 대해 배울 때 Alternate Functions에 대한 내용을 다룬 적이 있다. PortD의 5, 6번 Pin을 GPIO D 포트 기능 외에 추가적인 기능을 뜻한다. 그러나 위의 표에서 볼 수 있듯이 STM32f10x Chip의 USART 2번 채널의 Pin은 두 가지로 설정될 수 있다. 기본적으로 설정된 Pin은 PortA의 0~4번 Pin이며, USART2가 Remap 되었을 때 설정되는 Pin이 PortD의 3~7번 pin이다.

> *** 참고사항**
>
> 'Remap' 이란? CPU를 이용해서 어떤 작업을 수행할 때 CPU Package에서 제공되는 Pin을 이용하게 된다. 이 Pin들은 역할이 고정된 것도 있고, 또한 여러 가지 용도로 활용될 수 있는 핀들도 있다. 모든 핀이 고정된 역할이라면 다른 설정 없이 사용할 수 있으므로 편할 수 있지만, Pin의 개수가 많아져야 하므로 그만큼 가격이나 크기 역시 커지게 된다. 그래서 하나의 핀이 여러 역할을 할 수 있도록 하여 적은 개수의 Pin으로도 여러 동작을 할 수 있도록 설정하는 것을 Remap이라고 한다.

Remap을 설정하는 레지스터는 AFIO_MAPR 레지스터로, STM32에서는 GPIO_PinRemapConfig() 함수를 라이브러리 함수로 제공하고 있다. 레지스터와 함수에 대한 자세한 내용은 데이터 시트와 stm32f10x_gpio.c 파일을 참조한다.

실습 1. USART 제어 실습하기

USART 통신을 사용하여 하이퍼 터미널 화면에 문자열을 출력하고, 입력되는 값에 따라 LED를 켜고 끄는 소스 코드를 작성하라. ('1~4' 숫자 값을 입력받아 해당하는 LED가 켜지도록 하라.)

▣ Source Code (Project Name - USART_Ctrl) - main.c

```
#include <stm32f10x.h>

/* ========= GPIO 제어를 위한 정의 ========== */
#define   GPIO_LED                GPIOD
#define   RCC_LED                 RCC_APB2Periph_GPIOD
#define   LED1                    GPIO_Pin_7
#define   LED2                    GPIO_Pin_13
#define   LED3                    GPIO_Pin_14
#define   LED4                    GPIO_Pin_15
/* ===================================== */

/* ========= USART2 제어를 위한 정의 ========= */
#define   GPIO_USART              GPIOD
#define   RCC_AFIO                RCC_APB2Periph_AFIO
#define   RCC_USART               RCC_APB1Periph_USART2
#define   USART_TX_PIN            GPIO_Pin_5
#define   USART_RX_PIN            GPIO_Pin_6
/* ===================================== */

/* ================ 함수 선언 ============== */
void GPIO_Configuration();
void USART_Configuration();
void Serial_PutString(uint8_t*);
void SerialPutChar(uint8_t);
/* ===================================== */

/* ============ main 함수 구현 ============ */
int main()
{
        char strBuf[] = "Input Led Number : ";
        uint16_t rxData;

        GPIO_Configuration();
        USART_Configuration();

        while(1)
```

```
        {
                Serial_PutString(strBuf);       /* 문자열 출력 */

                /* Wait Receive Data */
                while(USART_GetFlagStatus(USART2, USART_FLAG_RXNE)==RESET) ;

                rxData = USART_ReceiveData(USART2);             /* Read Data */
                SerialPutChar(rxData);                          /* Output Data */

                /* LED All Off */
                GPIO_ResetBits(GPIO_LED, LED_ALL);

                /* Check Received Data & LED On */
                switch(rxData)
                {
                        case '1' :
                                GPIO_SetBits(GPIO_LED, LED1);
                                break;
                        case '2' :
                                GPIO_SetBits(GPIO_LED, LED2);
                                break;
                        case '3' :
                                GPIO_SetBits(GPIO_LED, LED3);
                                break;
                        case '4' :
                                GPIO_SetBits(GPIO_LED, LED4);
                                break;
                        default :
                                GPIO_ResetBits(GPIO_LED, LED_ALL);
                                break;
                }
                Serial_PutString("\r\n");       /* New Line */
        }
        return 0;
}

/* ========== GPIO 설정 함수 구현 ============ */
void GPIO_Configuration()
{
        GPIO_InitTypeDef GPIO_InitStruct;
```

```
        /* Clock Enable - GPIOD, AFIO for USART2 */
        RCC_APB2PeriphClockCmd(RCC_LED | RCC_AFIO, ENABLE);

        /* Configuration LED Pin */
        GPIO_InitStruct.GPIO_Pin = LED1 | LED2 | LED3 | LED4;
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIO_LED, &GPIO_InitStruct);

        /* Configuration USART2 Transmitter Pin */
        GPIO_InitStruct.GPIO_Pin = USART_TX_PIN;
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;            /* AFIO로 설정 */
        GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIO_USART, &GPIO_InitStruct);

        /* Configuration USART2 Receiver Pin */
        GPIO_InitStruct.GPIO_Pin = USART_RX_PIN;
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIO_USART, &GPIO_InitStruct);
}

/* ========== USART2 설정 함수 구현 ========= */
void USART_Configuration()
{
        USART_InitTypeDef USART_InitStruct;

        GPIO_PinRemapConfig(GPIO_Remap_USART2, ENABLE);     /* Set Remap Pin */
        RCC_APB1PeriphClockCmd(RCC_USART, ENABLE);          /* Enable Clock */

        USART_InitStruct.USART_BaudRate = 115200;           /* Set Baud Rate */
        USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
                                                            /* No Hardware Control */
        USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
                                                            /* Enable Transmitter */
        USART_InitStruct.USART_Parity = USART_Parity_No;    /* No Parity */
        USART_InitStruct.USART_StopBits = USART_StopBits_1; /* 1 Stop bit */
        USART_InitStruct.USART_WordLength = USART_WordLength_8b;     /* 8bit Data */

        USART_Init(USART2, &USART_InitStruct);
```

```
        /* Set USART_CR1 UE bit (USART Enable) */
        USART_Cmd(USART2, ENABLE);
}

void Serial_PutString(uint8_t* str)
{
        while(*str != '\0')
        {
                SerialPutChar(*str);
                str++;
        }
}

void SerialPutChar(uint8_t c)
{
        USART_SendData(USART2, c);
        while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET) ;
}
```

실습 5. Timer 제어하기

5.1 Timer 개요

STM32 Chip은 5가지 종류의 Timer를 제공한다.

첫 번째는 System Tick(SysTick) timer는 24 bit count-down timer로서 일정한 간격으로 interrupt를 만들어 준다. Real Time OS나 Scheduled Tasks를 구동하기 위해서 사용된다.

두 번째는 Watchdog이다. 시스템을 계속 쳐다보면서 감시하고 있다가 무한 Loop 발생 시에 Count-down 하여서 시스템을 Reset 한다. Watchdog은 IWDG(Independent Watchdog Timer), WWDG(Window Watchdog Timer)로 나뉜다. IWDG는 low-speed clock으로 내부 Clock을 사용하여 동작하는 것으로 Main Clock에 독립적으로 동작하며 낮은 정확도를 가진다. WWDG는 APB1 Clock을 Prescaler에 의해서 필요한 Clock을 얻어서 동작하는 것으로 S/W적인 오류를 검출해 주는 용도로 사용된다.

세 번째는 TIM 이름이 붙은 Timer이다. 3가지로 분류되며 아래와 같다.

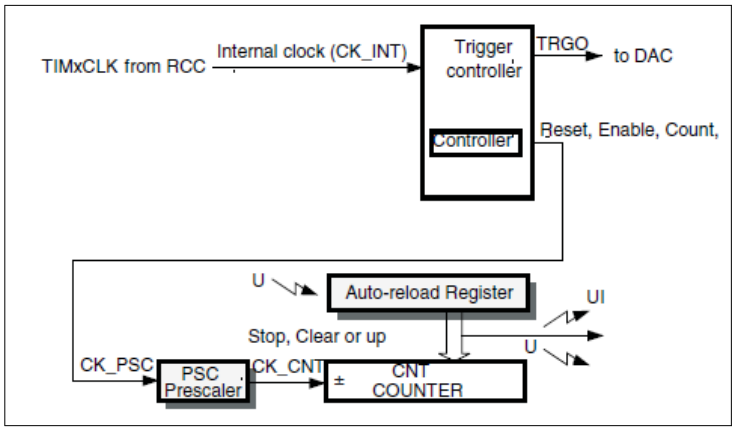
APB2를 사용하는 Advanced-Control Timer(TIM1, TIM8), APB1을 사용하는 General-purpose Timer TIMx(TIM2, TIM3, TIM4, TIM5)와 Basic Timer(TIM6, TIM7) 이다.

Timer들은 16-bit up, down, up/down auto-reload counter로 동작하며, Counter Clock 주파수를 분할하기 위해서 프로그램 Prescaler사용한다. 그리고 각 Timer들은 4개의 독립적인 channel로 구성되어 Input capture, Output compare, PWM generation (Edge- and Center-aligned modes), One-pulse mode output 기능을 수행한다.

Timer는 다음 3가지 핵심 레지스터로 동작을 제어한다.

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC)
- Auto_Reload Register(TIMx_ARR)

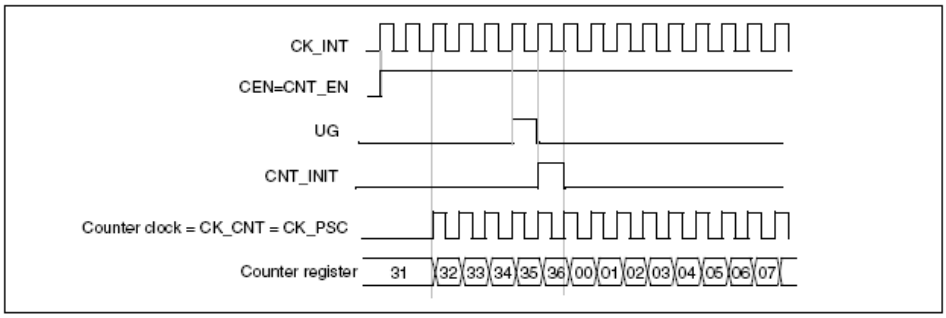
다음 그림은 Timer의 레지스터를 간단하게 나타낸 것이다.



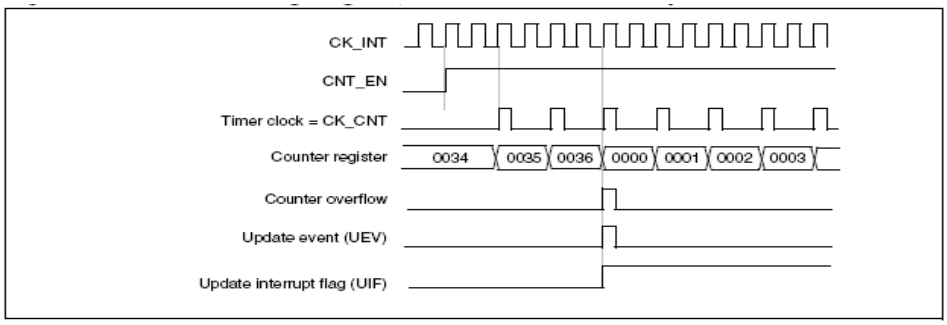
Counter Clock은 Internal Clock인 CK_INT에 의해 제공된다. TIMx_CR1 레지스터의 CEN bit와 TIMx_EGR 레지스터의 UG(Updata Generation) bit에 따라 제어되며, 그 값은 Software에 의해서만 변경될 수 있다.

다음은 Divide 값에 따른 CLK 타이밍도에 관한 그림이다.

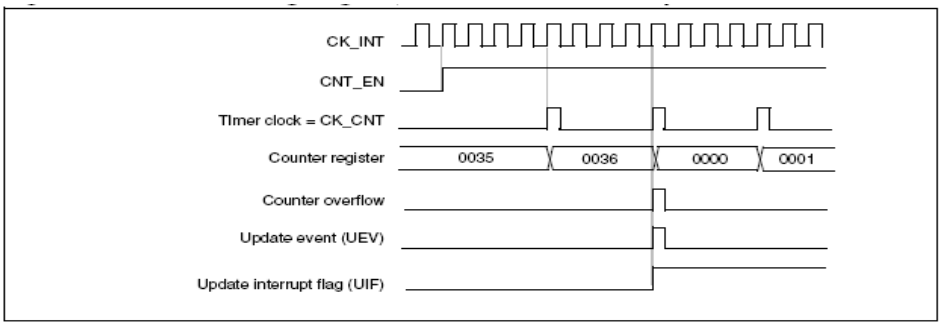
- Divided 1



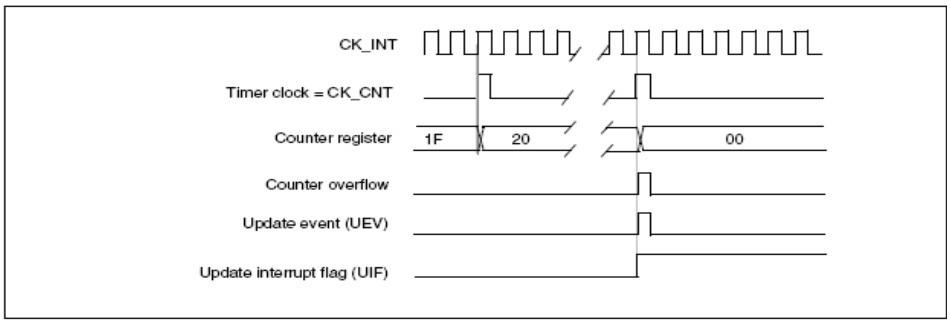
- Divided 2



- Divided 4

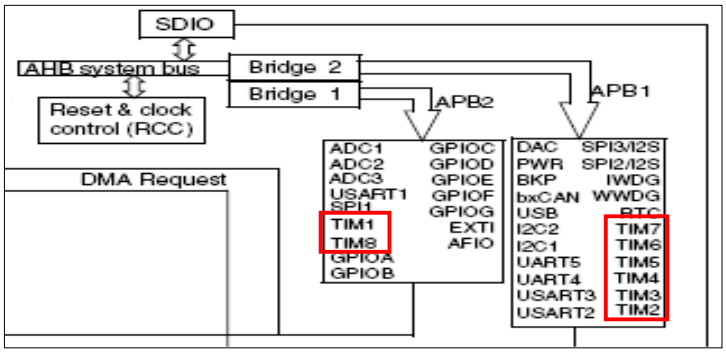


- Divided N



5.2 Timer 레지스터

각 Timer에 대한 레지스터 주소 영역 정보 및 동작시키는 clock 정보는 아래와 같다.



Boundary address	Peripheral	Bus
0x4001 3400 - 0x4001 37FF	TIM8 timer	APB2
0x4001 2C00 - 0x4001 2FFF	TIM1 timer	
0x4000 1400 - 0x4000 17FF	TIM7 timer	APB1
0x4000 1000 - 0x4000 13FF	TIM6 timer	
0x4000 0C00 - 0x4000 0FFF	TIM5 timer	
0x4000 0800 - 0x4000 0BFF	TIM4 timer	
0x4000 0400 - 0x4000 07FF	TIM3 timer	
0x4000 0000 - 0x4000 03FF	TIM2 timer	

Timer와 GPIO의 mapping 정보는 다음과 같다.

Advanced timers TIM1/TIM8

TIM1/8 pinout	configuration	GPIO configuration
TIM1/8_CHx	Input capture channel x	Input floating
	Output compare channel x	Alternate function push-pull
TIM1/8_CHxN	Complementary output channel x	Alternate function push-pull
TIM1/8_BKIN	Break input	Input floating
TIM1/8_ETR	External trigger timer input	Input floating

General-purpose timers TIM2/3/4/5

TIM2/3/4/5 pinout	Configuration	GPIO configuration
TIM2/3/4/5_CHx	Input capture channel x	Input floating
	Output compare channel x	Alternate function push-pull
TIM2/3/4/5_ETR	External trigger timer input	Input floating

상황에 따라서 지정된 pin이 아닌 다른 pin에 관련 기능을 설정해야 하는 경우 Re-mapping을 해야 한다. Timer관련 Remapping 정보는 다음과 같다.

TIM5 alternate function remapping

Alternate function	TIM5CH4_IREMAP = 0	TIM5CH4_IREMAP = 1
TIM5_CH4	TIM5 Channel4 is connected to PA3	LSI internal clock is connected to TIM5_CH4 input for calibration purpose.

TIM4 alternate function remapping

Alternate function	TIM4_REMAP = 0	TIM4_REMAP = 1(1)
TIM4_CH1	PB6	PD12
TIM4_CH2	PB7	PD13
TIM4_CH3	PB8	PD14
TIM4_CH4	PB9	PD15

TIM3 alternate function remapping

Alternate function	TIM3_REMAP[1:0] = "00" (no remap)	TIM3_REMAP[1:0] = "10" (partial remap)	TIM3_REMAP[1:0] = "11" (full remap) (1)
TIM3_CH1	PA6	PB4	PC6
TIM3_CH2	PA7	PB5	PC7
TIM3_CH3	PB0		PC8
TIM3_CH4	PB1		PC9

TIM2 alternate function remapping

Alternate function	TIM2_REMAP[1:0] = "00" (no remap)	TIM2_REMAP[1:0] = "01" (partial remap)	TIM2_REMAP[1:0] = "10" (partial remap) (1)	TIM2_REMAP[1:0] = "11" (full remap) (1)
TIM2_CH1_ETR(2)	PA0	PA15	PA0	PA15
TIM2_CH2	PA1	PB3	PA1	PB3
TIM2_CH3	PA2		PB10	
TIM2_CH4	PA3		PB11	

TIM1 alternate function remapping

Alternate functions mapping	TIM1_REMAP[1:0] = "00" (no remap)	TIM1_REMAP[1:0] = "01" (partial remap)	TIM1_REMAP[1:0] = "11" (full remap) (1)
TIM1_ETR	PA12		PE7
TIM1_CH1	PA8		PE9
TIM1_CH2	PA9		PE11
TIM1_CH3	PA10		PE13
TIM1_CH4	PA11		PE14
TIM1_BKIN	PB12 (2)	PA6	PE15
TIM1_CH1N	PB13	PA7	PE8
TIM1_CH2N	PB14 (2)	PB0	PE10
TIM1_CH3N	PB15 (2)	PB1	PE12

Timer에서 사용되는 Register에 대해서 살펴보자.

1. TIMx_CR1 (TIMx Control Register 1)

Address offset : 0x00, Reset Value : 0x00000000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

레지스터의 각 bit는 Timer 상태 정보를 나타내고 있는데, 여기서는 실습에 필요한 부분만 살펴보도록 하겠다.

- **CKD** (Clock Division) : CK_INT(Timer clock)과 Sampling clock(ETR, TIx) 사이의 분배율을 설정한다.
  00 : tDTS = tCK_INT
  01 : tDTS = 2 * tCK_INT
  10 : tDTS = 4 * tCK_INT
  11 : Reserved
- **DIR** (Direction Register) : counter 증가 방향을 설정한다. Center-aligned mode 또는 Encoder mode가 설정되었을 경우 읽혀지는 bit이다.
  0 : up counter
  1 : down counter
- **CMS** (Center-aligned Mode Selection) : Center Aligned mode를 선택한다.
  00 : Edge-aligned mode
  01 : tDTS = 2 * tCK_INT
  10 : tDTS = 4 * tCK_INT
  11 : Reserved
- **CEN**(Counter Enable) : '1' 로 설정되면 Timer를 활성화 시킨다. Update Event 발생했을 때 하드웨어에 의해 자동으로 Clear된다.

2. TIMx_DIER (TIMx DMA/Interrupt Enable Register)

Address offset : 0x0C, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Update event가 발생할 수 있도록 활성화 시키는 레지스터이다.

- **UE** (Update Interrupt Enable) : Update event가 발생할 수 있도록 Update Interrupt를 활성화 시킨다.

3. TIMx_SR (TIMx Status Register)

Address offset : 0x10, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

TIM 관련 다양한 상태 정보를 설정하는 레지스터이다.

- **UIF** (Update Interrupt Flag)
  0 : No update occurred. 반드시 S/W Interrupt Handler에서 '0' 으로 clear 해야 한다.
  1 : Update interrupt pending. update event가 발생하면 H/W에서 '1' 로 Set 된다.

4. TIMx_EGR (TIMx Event Generation Register)

Address offset : 0x14, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									TG	Res.	CC4G	CC3G	CC2G	CC1G	UG
									w		w	w	w	w	w

- **UG** (Update Generation) : Counter에 대한 재 초기화 작업 수행한다. 이 bit가 '1' 로 설정되면 Center-aligned mode 또는 Up-counting mode 일 경우 Counter를 '0' 으로 Reset하며 Down-counting mode일 경우 TIMx_ARR 값으로 Counter를 Set 한다.

5. TIMx_CCMRn (TIMx Capture/Compare Mode Register n) (n : 1..2)

Address offset : 0x18~0x1C, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Timer의 1~4번 채널을 설정하는 레지스터이다. TIMx_CCMR1 레지스터는 1, 2번 채널을, TIMx__CCMR2 레지스터는 3,4번 채널을 설정한다.

- **OCnCE** (Output Compare n Clear Enable)
- **OCnM[2:0]** (Output Compare n Mode)
- **OCnPE** (Output Compare n Preload Enable)
- **OCnFE** (Output Compare n Fast Enable)
- **CCnS[1:0]** (Capture/Compare n Selection) : Channel의 Input/Output mode를 설정한다.
  00 : CCn channel을 output 설정
  01 : CCn channel을 input 설정 / ICn은 TI2에 Mapping
  10 : CCn channel을 output 설정 / ICn은 TI1에 Mapping
  11 : CCn channel을 input 설정 / ICn은 TRC에 Mapping

7. TIMx_CCER (TIMx Capture/Compare Enable Register)

Address offset : 0x20, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		CC4P	CC4E	Reserved		CC3P	CC3E	Reserved		CC2P	CC2E	Reserved		CC1P	CC1E
		rw	rw			rw	rw			rw	rw			rw	rw

Output Enable, Output Polarity(극성)를 설정하는 레지스터이다. 각 bit의 설정 값은 다음과 같다.

- **CCnE** (Capture/Compare n output enable) : CCn channel이 Output mode일 경우 OCn을 활성화 시키며, CCn이 Input mode일 경우 Capture 기능을 활성화 시킨다.

8. TIMx_CNT (TIMx Counter)

Address offset : 0x24, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **CNT[15:0]:** Counter 값이 저장된다. 이 값과 TIMx_ARR의 값이 일치되고 다시 '0' 이 되는 순간에 Update Event가 발생한다.

9. TIMx_PSC (TIMx Prescaler)

Address offset : 0x28, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **PSC[15:0]** (Prescaler) : 분주비를 설정한다. 각 Update Event마다 Prescaler Register에 값이 Set 된다.

10. TIMx_ARR (TIMx Auto-Reload Register)

Address offset : 0x2C, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **ARR[15:0]** : Counter 값으로 자동 재설정될 때 적용될 값을 설정한다.

11. TIMx_CCRn (TIMx Capture/Compare Register n) (n : 1..4)

Address offset : 0x34~0x40, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

n번 채널의 값을 설정한다. Output mode에서 Preload된 값으로 TIMx_CCMR의 OCxPE가 설정된 경우 Update Event 시에 값이 갱신되며, OCxPE가 설정되지 않은 경우에는 즉시 값이 갱신된다. Input mode에서는 Input Capture n Event(ICn)을 통해 전송된 Counter 값이 저장된다.

5.3 Timer 라이브러리 함수

Timer 제어를 위해 필요한 라이브러리 함수에 대해서 알아보자.

1. TIM_TimerBaseInit

void **TIM_TimeBaseInit**(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)	
▪ 기능	Timer의 설정 값을 초기화한다.
▪ 인수	TIMx - 설정할 Timer 채널 Tim_TimeBaseInitStruct - 해당 채널의 초기 설정 값을 담고 있는 구조체
▪ stm32f10x_tim.c 참고	

TIM_TypeDef 구조체는 TIM관련 register를 선언한 것으로 다음과 같다.

```
typedef struct
{
  __IO uint16_t CR1;
  uint16_t  RESERVED0;
 __IO uint16_t CR2;
  uint16_t  RESERVED1;
  __IO uint16_t SMCR;
  uint16_t  RESERVED2;
 __IO uint16_t DIER;
  uint16_t  RESERVED3;
  __IO uint16_t SR;
  uint16_t  RESERVED4;
 __IO uint16_t EGR;
  uint16_t  RESERVED5;
  __IO uint16_t CCMR1;
  uint16_t  RESERVED6;
 __IO uint16_t CCMR2;
  uint16_t  RESERVED7;
  __IO uint16_t CCER;
  uint16_t  RESERVED8;
 __IO uint16_t CNT;
  uint16_t  RESERVED9;
```

```
  __IO uint16_t PSC;
  uint16_t  RESERVED10;
 __IO uint16_t ARR;
  uint16_t  RESERVED11;
__IO uint16_t RCR;
  uint16_t  RESERVED12;
 __IO uint16_t CCR1;
  uint16_t  RESERVED13;
  __IO uint16_t CCR2;
  uint16_t  RESERVED14;
 __IO uint16_t CCR3;
  uint16_t  RESERVED15;
  __IO uint16_t CCR4;
  uint16_t  RESERVED16;
 __IO uint16_t BDTR;
  uint16_t  RESERVED17;
  __IO uint16_t DCR;
  uint16_t  RESERVED18;
 __IO uint16_t DMAR;
  uint16_t  RESERVED19;
} TIM_TypeDef
```

TIM_TimeBaseInitTypeDef 구조체의 원형은 다음과 같다.

```
typedef struct
{
  uint16_t TIM_Prescaler
  uint16_t TIM_CounterMode
  uint16_t TIM_Period
  uint16_t TIM_ClockDivision
  uint16_t TIM_RepetitionCounter
} TIM_TimeBaseInitTypeDef
```

2. TIM_Cmd

void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)	
▪ 기능	Timer를 활성화 시킨다. (TIMx_CR1의 CEN bit 활성화)
▪ 인수	TIMx - Timer 채널 NewState - 활성화 여부 (ENABLE or DISABLE)
▪ stm32f10x_tim.c 참고	

3. TIM_ClearITPendingBit

void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, uint16_t TIM_IT)	
▪ 기능	TIMx_SR 레지스터의 interrupt pending bit를 '0'으로 Clear 한다.
▪ 인수	TIMx - Timer 채널 TIM_IT - 0으로 clear할 interrupt bit
▪ stm32f10x_tim.c 참고	

4. TIM_PrescalerConfig

void TIM_PrescalerConfig(TIM_TypeDef* TIMx, uint16_t Prescaler, uint16_t TIM_PSCReloadMode)	
▪ 기능	Timer Update Event 처리를 위한 설정
▪ 인수	TIMx - Timer 채널 Prescaler - CK_CNT 값 계산에 사용되는 값, TIMx_PSC에 저장 TIM_PSCReloadMode - TIMx_EGR의 UG 비트 설정, counter에 대한 재 초기화 관련 설정 0: 아무 영향 안 끼침 1: down-counting 경우 TIMx_ARR 으로 설정 그 외의 counting 경우 0으로 설정
▪ stm32f10x_tim.c 참고	

5. TIM_ITConfig

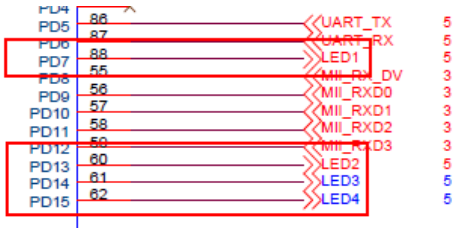
void **TIM_ITConfig**(TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState NewState)	
▪ 기능	TIMx_DIER의 UIE 비트 설정으로 Update Interrupt를 Enable
▪ 인수	TIMx - Timer TIM_IT- TIM Interrupt 발생 조건 값 TIM_IT_Update / TIM_IT_CC1 ~CC4/ TIM_IT_COM NewState - TIMx_DIER의 UIG 비트 설정
▪ stm32f10x_tim.c 참고	

6. TIM_ClearFlag

void **TIM_ClearFlag**(TIM_TypeDef* TIMx, uint16_t TIM_FLAG)	
▪ 기능	TIMx_SR의 특정 Flag를 clear 함
▪ 인수	TIMx - Timer TIM_FLAG -TIM Interrupt Flag 및 Capture/Compare overcapture flag
▪ stm32f10x_tim.c 참고	

5.4 Timer 제어 실습하기

Timer 제어 실습을 위해서 LED를 사용하도록 하겠다. 1초 간격으로 LED가 켜지도록 한다. 다음 그림은 실습 보드에서 LED가 연결된 모습을 나타낸 것이다.



실습 1. Timer 제어 실습하기

Timer를 사용하여 1초 간격으로 LED를 켜고 끄는 소스 코드를 작성하라.

* 헤더 파일 추가

▣ Source Code (Project Name - Timer_LED) - Timer_LED.h

```
#ifndef __TIMER_LED_H__
#define __TIMER_LED_H__

#include "stm32f10x.h"
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

/* ========= USART2 제어를 위한 정의 ========== */
#define   GPIO_USART          GPIOD
#define   RCC_AFIO            RCC_APB2Periph_AFIO
#define   RCC_USART           RCC_APB1Periph_USART2
#define   USART_TX_PIN        GPIO_Pin_5
#define   USART_RX_PIN        GPIO_Pin_6


/* =========== LED 제어를 위한 정의 =========== */
#define   GPIO_LED            GPIOD
#define   RCC_LED             RCC_APB2Periph_GPIOD
#define   LED1                GPIO_Pin_7
```

```
/* ========== TIM 제어를 위한 정의 ============ */
#define   RCC_TIM                    RCC_APB1Periph_TIM2

/* ========== LED Function  ============ */
void LED_Toggle(void);

/* ========== USART Function  ========== */
void USART_Configuration();
void SerialPutChar(uint8_t c);
void Serial_PutString(uint8_t* str);
void Uart_Printf(const char *fmt,...);

/* ========= Private function ======== */
void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void Timer_Configuration(void);

#endif /* __TIMER_LED_H__ */
```

* Timer_LED.c 파일 추가

▣ Source Code (Project Name - Timer_LED) - Timer_LED.c

```
#include "stm32f10x.h"
#include "Timer_LED.h"

/* ========== USART  설정 함수 구현 ========== */
/* USART TX, RX 설정   */
void USART_Configuration()
{
        USART_InitTypeDef USART_InitStruct;

        GPIO_PinRemapConfig(GPIO_Remap_USART2, ENABLE);     /* Set Remap Pin */
        RCC_APB1PeriphClockCmd(RCC_USART, ENABLE);          /* Enable Clock */

        USART_InitStruct.USART_BaudRate = 115200;           /* Set Baud Rate */
        /* No Hardware Control */
        USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
        /* Enable Transmitter */
        USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
        USART_InitStruct.USART_Parity = USART_Parity_No;                /* No Parity */
```

```
        USART_InitStruct.USART_StopBits = USART_StopBits_1;          /* 1 Stop bit */
        USART_InitStruct.USART_WordLength = USART_WordLength_8b;     /* 8bit Data */
        USART_Init(USART2, &USART_InitStruct);

        /* Set USART_CR1 UE bit (USART Enable) */
        USART_Cmd(USART2, ENABLE);
}

/* ========== USART 문자 출력 함수 구현 ========== */
void SerialPutChar(uint8_t c)
{
        USART_SendData(USART2, c);
        while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET) ;
}

/* ========== USART 문자열 출력 함수 구현 ========== */
void Serial_PutString(uint8_t* str)
{
        while(*str != '\0')
        {
                SerialPutChar(*str);
                str++;
        }
}

/* ========== USART 표준 출력 함수 구현 ========== */
void Uart_Printf(const char *fmt,...)
{
    va_list ap;
    char string[256];

    va_start(ap,fmt);
    vsprintf(string,fmt,ap);
    Serial_PutString(string);
    va_end(ap);
}

/* ========== LED on/off 토글링 함수 구현 ========== */
void LED_Toggle(void)
{
```

```
    GPIO_LED->ODR ^= LED1;
    Uart_Printf("TIM2_IRQHandler()-0x%x\n", GPIOD->ODR);
}

/* ========== RCC_Configuration 함수 구현 ========== */
/* LED, TIM2 Device clock enable/disable  설정  */
void RCC_Configuration(void){

        /* Enable the LED Clock */
        RCC_APB2PeriphClockCmd(RCC_LED | RCC_AFIO, ENABLE);

        /* Enable the TIM2 Clock */
        RCC_APB1PeriphClockCmd(RCC_TIM, ENABLE);
}

/* ========== NVIC_Configuration함수 구현 ========== */
/* TIM2 Interrupt 등록, 처리 위한 NVIC 설정  */
void NVIC_Configuration(void)
{
        NVIC_InitTypeDef NVIC_InitStructure;

        NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

        /* Enable the TIM2 global Interrupt */
        /* TIM2 Interrupt 발생 시  TIM2_IRQHandler() 호출*/
        NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);
}

/* ========== GPIO_Configuration함수 구현 ========== */
/* LED, USART TX, RX GPIO 설정  */
void GPIO_Configuration(void)
{
        GPIO_InitTypeDef   GPIO_InitStruct;

        /*-----------LED GPIO -----------*/
```

```
        GPIO_InitStruct.GPIO_Pin      = LED1;
        GPIO_InitStruct.GPIO_Speed  = GPIO_Speed_50MHz;
        GPIO_InitStruct.GPIO_Mode    = GPIO_Mode_Out_PP;
        GPIO_Init(GPIOD, &GPIO_InitStruct);

        /* Configuration USART2 Transmitter Pin */
        GPIO_InitStruct.GPIO_Pin      = USART_TX_PIN;
        GPIO_InitStruct.GPIO_Mode    = GPIO_Mode_AF_PP;          /* AFIO로 설정 */
        GPIO_InitStruct.GPIO_Speed   = GPIO_Speed_50MHz;
        GPIO_Init(GPIO_USART, &GPIO_InitStruct);

        /* Configuration USART2 Receiver Pin */
        GPIO_InitStruct.GPIO_Pin      = USART_RX_PIN;
        GPIO_InitStruct.GPIO_Mode    = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIO_USART, &GPIO_InitStruct);
}

/* ==========Timer Configuration함수 구현 ========== */
/* Timer 설정  */
void Timer_Configuration(void)
{
          TIM_TimeBaseInitTypeDef   TIM_TimeBaseStructure;

          /* Time base configuration */
          TIM_TimeBaseStructure.TIM_Period = 0x4AF; //65535;
          TIM_TimeBaseStructure.TIM_Prescaler = 0xEA5F;
          TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
          TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

          /* TIM2 설정 */
          TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

          /* TIM2 enable counter */
          TIM_Cmd(TIM2, ENABLE);

          /* Prescaler configuration */
          TIM_PrescalerConfig(TIM2, 0xEA5F, TIM_PSCReloadMode_Immediate);

          /* TIM2의  TIMx_SR의 interrupt flag 0 으로 set, pending bit clear */
          TIM_ClearFlag(TIM2, TIM_FLAG_Update);
```

```
        /* TIM IT enable */
        TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
}
```

* 메인 함수 구현

▣ Source Code (Project Name - Timer_LED) - main.c

```
#include "stm32f10x.h"
#include "Timer_LED.h"

/* ========== Main 함수 구현 ========== */
int main(void)
{
        /* Clock Enable - LED,  TIM2 */
        RCC_Configuration();

        /* NVIC Configuration - TIM2 */
        NVIC_Configuration();

        /*  Setting - LED / USART GPIO  */
        GPIO_Configuration();
        USART_Configuration();

        /*  Setting - TIM2  */
        Timer_Configuration();

        while(1)
        {
                ;       /* Loop Wait Interrupt */
        }
        return 0;
}
```

* 인터럽트 처리 함수 구현

▣ Source Code (Project Name - Timer_LED) - stm32f10x_it.c

```
/* ========== TIM2 Interrupt 함수 구현 ========== */
void TIM2_IRQHandler(void)
{
        /* Clear TIM2 update interrupt*/
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);

        /* LED On/Off */
        LED_Toggle();
}
```

실습 6. SPI 제어하기

6.1 SPI(Serial Peripheral Interface) 개요

SPI(Serial Peripheral Interface)란 쉽게 말해 직렬 통신을 말한다. 직렬 통신으로는 UART, I2C, USB 등이 있으며, 통상적으로 SPI는 Clock, Data Line을 이용한 동기식 직렬 통신을 의미한다. 여기에 Stobe(또는 Chip Selector)를 하나 더 붙이면 3선식 직렬 통신 방식이 되며, Data Line을 Data In, Out으로 구분하면 4선식 직렬 통신 방식이 된다.

SPI는 Socket 통신의 Server/Client 모델과 유사하게 Master/Slave 모델을 사용한다. Master는 SPI 통신 전체를 관리한다. SPI 통신은 두 장치의 데이터 전송이 동시에 이루어지기 때문에 데이터 송신 시 바로 데이터 수신이 되며, 전송 타이밍을 결정하는 것은 마스터이다.

SPI는 기본적으로 3pin 혹은 4pin과 shift register로 구성되며 pin 구성은 다음과 같다.

- MOSI (Master Out Slaver In) : 마스터가 생성하고 슬레이브가 수신
- MISO (Mater In Slaver Out) : 슬레이브에 의해서 생성되고 마스터에서 제어
- SCK (Serial Clock) : 데이터 전송을 하는 타이밍을 Master에서 Slave로 제공

SPI 통신에서 Master는 무조건 하나여야 하지만, Slave는 여러 대가 될 수 있으며, 이런 경우를 구분하기 위해 다음과 같은 방법을 사용한다.

- Slave 마다 CS(Chip Select)선 추가 사용
- Slave 마다 SID(Slave ID) 부여
- Demultiplexer 사용

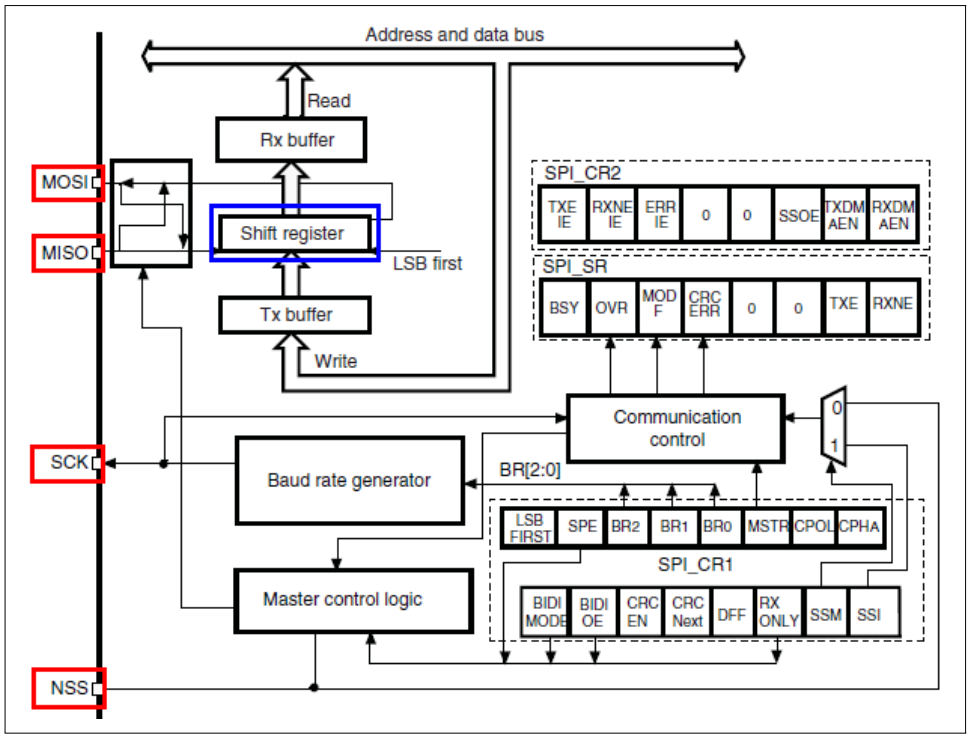
이 중에서 CS선을 사용하는 방법 이외의 사용은 매우 드물다.

SPI는 장치의 수가 적을 때 유용한 통신방식이다. 장치가 많아질수록 추가적인 부담이 늘어나며, 만약 8개가 넘어간다면 SPI 통신 이외에 다른 통신 방식을 찾아보는 것이 더 좋다.

지금까지 SPI에 대해 설명을 했다. 이제부터 우리가 사용하는 실습 장비인 STM32의 SPI에 대해서 살펴보자.

STM32 Chip에서는 최소 1개 이상의 SPI 통신 기능을 제공한다. 고성능의 STM32 IC는 SPI와 I2S Protocol을 기본적으로 제공하며 소프트웨어적으로 SPI를 I2S로 전환 할 수 있도록 한다. 그 외의 STM32 Chip에서는 I2S Protocol을 지원하지 않는다.

다음 그림은 SPI 통신을 그림으로 나타낸 것이다.

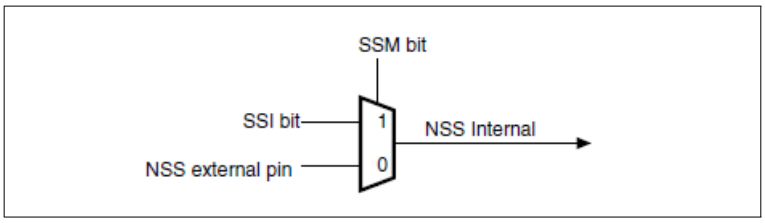


SPI와 외부 장치를 연결하기 위해서 4 Pin이 존재한다. 그리고 master와 slave가 통신하기 위한 shift register가 존재한다. 각 pin의 역할은 다음과 같다.

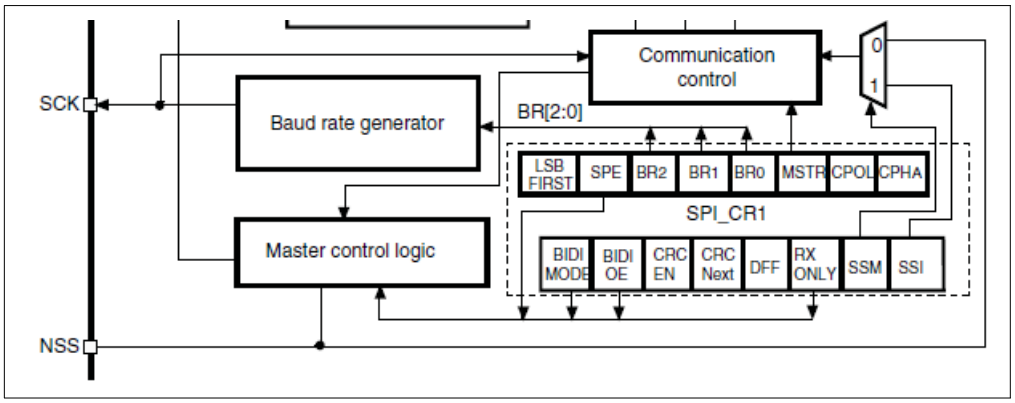
- MOSI (Master Out Slaver In) : master 생성/전송 --> slave 수신
- MISO (Mater In Slaver Out) : slave 생성/전송 --> master 수신
- SCK (Serial Clock) : master clock output --> slave clock input
- NSS (Negative Slave Select) : slave 장치를 선택하기 위한 옵션 pin. 'chip select'로 동작

NSS에 대해서 좀 더 살펴보자. NSS는 두 가지 모드가 존재한다.

- Software NSS mode : Chip Select Pin으로 사용하지 않는다. Master와 Slave가 1:1인 경우 Slave 선택이 필요 없으므로, SPI_CR1 레지스터의 SSM bit를 설정하는데 사용된다.
- Hardware NSS mode : 1개의 Master와 여러 개의 Slave로 구성된 경우 특정 Slave 선택을 위해 사용된다.
  - NSS output is enabled : SPI_CR2의 SSOE bit set, 연결된 모든 device들은 low에 동작
  - NSS output is disabled : 다중 master 능력 허용



SPI의 Clock 생성에 관해 살펴보자.



Master는 Clock을 생성해서 Slave로 보내기 위해서 SCK Pin으로 출력한다. SPI_CR1 Register의 CPHA, CPOL bit를 사용해서 software적으로 Clock을 설정할 수 있다. CPOL(Clock POLarity) bit는 데이터가 전송되지 않을 때 Clock이 정상적인 값이 되도록 제어 한다. 이것은 Master와 Slave 모두에게 영향을 끼친다. 만약 CPOL이 Reset되면 SCK Pin은 Low-Level Idle 상태가 된다. CPOL이 Set되면 SCK Pin은 High-Level Idle 상태가 된다.

만약, CPHA bit가 Set 되었다면, SCK의 두 번째 Edge에 MSBit가 감지되고, 두 번째 Clock이 변형될 때 데이터를 읽어온다. 만약 CPHA bit가 Reset되었다면, SCK의 첫 번째 Edge에 MSBit가 감지되고, 첫 번째 Clock이 변형될 때 데이터가 읽어온다.

이 두 bit를 조합해서 데이터 감지 Clock Edge를 선택한다.

Master/Slave SPI 사이의 송수신을 위해서 Data Frame Format을 정해야한다. SPI_CR1 레지스터의 DFF bit에 지정된 크기에 따라 8bit 또는 16bit의 Data Frame으로 결정한다. 그리고 지정된 데이터를 전송 시에 상위 bit(MSB-first)부터 송수신할지, 하위 bit(LSB-first)부터 송수신할지 지정해야 한다.

이것은 SPI_CR1의 LSBFIRST bit에 설정한다.

SCK 설정을 위한 순서는 다음과 같다.

(1) SPI_CR1의 SPE bit를 disable 시킨다.

(2) Master/Slave는 같은 Timing mode 설정해야 한다.

(3) SCK의 Idle State는 SPI_CR1의 CPOL bit 값에 따라서 설정된다.

CPOL bit가 '1' 이면 Pulling Up

CPOL bit가 '0' 이면 Pulling Down

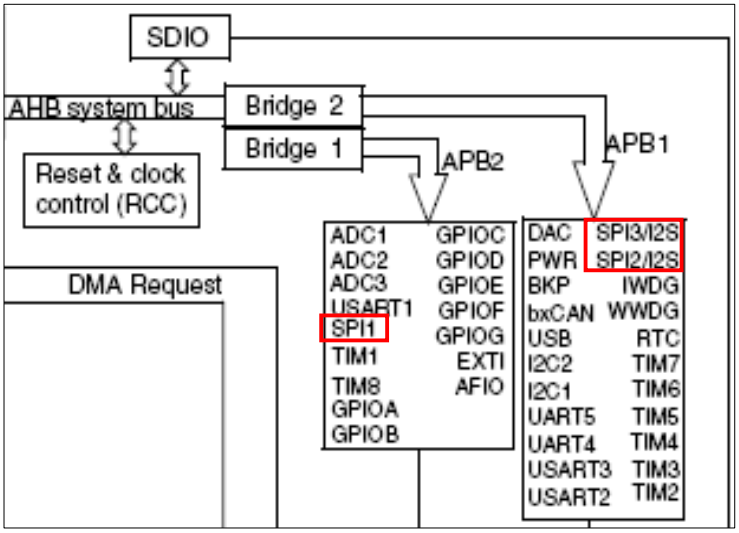
(4) Data Frame Format 설정한다. SPI_CR1의 DFF bit에 따라서 Format과 송수신 동안의 Data Length가 결정된다.

6.2 SPI 레지스터

SPI 제어를 위한 Register에 대해서 살펴보도록 한다.

STM32는 3개의 SPI를 제공한다. 그 중 SPI2/SPI3은 I2S와 함께 사용이 되고 있다.

Boundary address	Peripheral	Bus
0x4001 3000 - 0x4001 33FF	SPI1	APB2
0x4000 3C00 - 0x4000 3FFF	SPI3/I2S	APB1
0x4000 3800 - 0x4000 3BFF	SPI2/I2S	



SPI 레지스터 맵과 GPIO 관계는 다음과 같다.

SPI1 remapping

Alternate function	SPI1_REMAP = 0	SPI1_REMAP = 1
SPI1_NSS	PA4	PA15
SPI1_SCK	PA5	PB3
SPI1_MISO	PA6	PB4
SPI1_MOSI	PA7	PB5

SPI3 remapping

Alternate function	SPI3_REMAP = 0	SPI3_REMAP = 1
SPI3_NSS	PA15	PA4
SPI3_SCK	PB3	PC10
SPI3_MISO	PB4	PC11
SPI3_MOSI	PB5	PC12

SPI와 Interrupt관계는 다음과 같다.

Vector table for connectivity line devices

Position	Priority	Type of priority	Acronym	Description	Address
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
51	58	settable	SPI3	SPI3 global interrupt	0x0000_010C

1. SPI_CR1 (SPI Control Register 1)

Address offset : 0x00, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

레지스터의 각 bit는 SPI 통신 상태 정보를 나타내고 있는데, 여기서는 실습에 필요한 부분만 살펴보도록 하겠다. 이 레지스터는 SPI의 기본적인 제어를 위한 것으로 SCK Clock 생성 관련, 송수신 Data Frame Format 관련 설정을 한다.

- **CPHA** (Clock Phase) : 데이터 감지하기 위한 Edge 선택한다.
  0 : 첫 번째 clock transition
  1 : 두 번째 clock transition
- **CPOL** (Clock Polarity) : Idle 상태일 때 Clock 값, 시작하는 Clock 값이 설정된다.
  0 : LOW
  1 : High
- **LSBFIRST** (Frame Format) : 수신 시 데이터 전송 시작 bit 설정한다.
  0 : MSB transmitted first
  1 : LSB transmitted first
- **SPE** (SPI Enable) : '1' 로 설정되면 SPI를 활성화 시킨다.
- **SSM** (Software Slave Management) : SSM bit가 set 되면 SSI bit의 값이 설정된다.
  0 : Software slave management disabled
  1 : Software slave management enabled
- **SSI** (Internal Slave Select) : SSM bit가 Set된 경우만 효과를 가진다. NSS pin의 I/O 값은 무시된다.
- **DFF** (Data Frame Format)
  0 : 8-bit data frame format is selected for transmission/reception
  1 : 16-bit data frame format is selected for transmission/reception

2. SPI_CR2 (SPI Control Register 2)

Address offset : 0x04, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	Reserved		SSOE	TXDMAEN	RXDMAEN
								rw	rw	rw			rw	rw	rw

- **TXEIE** (Tx Buffer Empty Interrupt Enable)
  0 : TXE interrupt masked
  1 : TXE interrupt not masked. TXE Flag가 Set되어 있을 경우 인터럽트를 요청한다.
- **RXNEIE** (RX Buffer Not Empty Interrupt Enable)
  0 : RXNE interrupt masked
  1 : RXNE interrupt not masked. RXNE Flag가 Set되어 있을 경우 인터럽트를 요청한다.
- **SSOE** (SS Output Enable)
  0 : Master Mode에서 SS Output을 Disabled. Multi-Master Configuration.
  1 : Master Mode에서 SS Output을 Enabled. Multi-Master 구성 불가.

3. SPI_SR (SPI Status Register)

Address offset : 0x08, Reset Value : 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BSY	OVR	MODF	CRC ERR	UDR	CHSIDE	TXE	RXNE
								r	r	r	rc_w0	r	r	r	r

레지스터의 각 bit는 SPI 통신 상태 정보를 나타내고 있는데, 여기서는 실습에 필요한 부분만 살펴보도록 하겠다

- **BSY** (Busy Flag) : 하드웨어에 의해서 Set/Clear 된다.
  0 : SPI (or I2S) not busy
  1 : SPI (or I2S) is busy in communication or Tx buffer is not empty
- **OVR** (Overrun Flag) : 하드웨어에 의해서 Set되고, 소프트웨어에서 Reset 된다.
  0 : No overrun occurred
  1 : Overrun occurred
- **TXE** (Transmit Buffer Empty)
  0 : Tx buffer not empty
  1 : Tx buffer empty
- **RXNE**(Receive Buffer Not Empty)
  0 : Rx buffer empty
  1 : Rx buffer not empty

4. SPI_DR (SPI Data Register)

Address offset : 0x0C, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **DR[15:0]** (Data Register) : 송신 버퍼와 수신 버퍼로 구성된 데이터 저장 레지스터이다. Data Register에서 Tx Buffer에 데이터를 기록하고, Rx Buffer에서 반환한 값을 Data Register로부터 읽어온다. 데이터 읽기/쓰기는 SP1_CR1의 DFF로 Data Frame을 설정한다.

8 bit Data Frame - 8bit 버퍼 사용
SPI_DR[7:0]의 데이터를 LSB 모드로 송수신
수신 모드 일 경우, SPI_DR[15:8]의 데이터를 '0' 으로 설정

16 bit Data Frame - 16bit 버퍼 사용
SPI_DR[15:0]의 전체 데이터를 송수신

7. SPI_I2SCFGR (SPI_I2S Configuration Register)

Address offset : 0x1C, Reset Value : 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				I2SMOD	I2SE	I2SCFG		PCMSYNC	Reserved	I2SSTD		CKPOL	DATLEN		CHLEN
				rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw

- **I2SMOD** (I2S Mode Selection)
  0 : SPI mode is selected
  1 : I2S mode is selected
- **I2SE** (I2S Enable) : I2S를 활성화 시킨다. SPI 모드에서는 사용하지 않는다.
  0 : I2S peripheral is disabled
  1 : I2S peripheral is enabled
- **I2SCFG** (I2S Configuration Mode)
  00 : Slave - Transmit
  01 : Slave - Receive
  10 : Master - Transmit
  11 : Master - Receive

6.3 SPI 라이브러리 함수

SPI 제어를 위해 필요한 라이브러리 함수에 대해서 알아보자.

1. SPI_Init

void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)	
▪ 기능	USART의 해당 채널을 설정 값을 초기화한다.
▪ 인수	SPIx - 설정할 SPI SPI_InitStruct - SPI 관련 설정 값을 담고 있는 초기화 구조체 포인터
▪ stm32f10x_spi.c 참고	

SPI_InitTypeDef 구조체의 원형은 다음과 같다.

```
typedef struct
{
  uint16_t SPI_Direction;
  uint16_t SPI_Mode;
  uint16_t SPI_DataSize;
  uint16_t SPI_CPOL;
  uint16_t SPI_CPHA;
  uint16_t SPI_NSS;
  uint16_t SPI_BaudRatePrescaler;
  uint16_t SPI_FirstBit;
  uint16_t SPI_CRCPolynomial;
} USART_InitTypeDef
```

SPI_Direction bit에 송수신 기능, 수신 기능, 송신 기능 지정 및 in/out pin의 통합 및 분리 등을 설정하며, 다음과 같이 정의 되어 있다.

```
#define SPI_Direction_2Lines_FullDuplex ((uint16_t)0x0000)
#define SPI_Direction_2Lines_RxOnly     ((uint16_t)0x0400)
#define SPI_Direction_1Line_Rx          ((uint16_t)0x8000)
#define SPI_Direction_1Line_Tx          ((uint16_t)0xC000)
```

SPI_CPOL, CPHA bit를 사용해서 SCK Clock을 설정하고, 데이터를 검출할 Clock Edge를 설정한다. SPI_Mode bit는 Master 인지 Slave인지 설정한다. SPI_DataSize bit는 송수신에 사용될 Data Frame Format을 설정하며, SPI_FirstBit bit에 송수신 시 데이터의 시작 bit를 설정한다.

2. SPI_Cmd

void **SPI_Cmd**(SPI_TypeDef* SPIx, FunctionalState NewState);)	
▪ 기능	SPI를 활성화 시킨다. (SPI_CR1의 SPE bit 활성화)
▪ 인수	SPIx- 활성화 시킬 SPI NewState - 활성화 여부 (ENABLE or DISABLE)
▪ stm32f10x_spi.c 참고	

3. SPI_SendData

void **SPI_I2S_SendData**(SPI_TypeDef* SPIx, uint16_t Data)	
▪ 기능	SPI의 해당 채널로 데이터를 전송한다.
▪ 인수	SPIx- 데이터를 전송할 SPIx 채널 Data - 전송할 데이터
▪ stm32f10x_spi.c 참고	

4. SPI_I2S_ReceiveData

uint16_t **SPI_I2S_ReceiveData**(SPI_TypeDef* SPIx)	
▪ 기능	SPI의 해당 채널에서 데이터를 수신
▪ 인수	SPIx- 데이터를 수신할 SPIx 채널
▪ 반환 값	uint16_t - 수신한 데이터
▪ stm32f10x_spi.c 참고	

5. SPI_I2S_GetFlagStatus

FlagStatus **SPI_I2S_GetFlagStatus**(SPI_TypeDef* SPIx, uint16_t SPI_I2S_FLAG)	
▪ 기능	SPI의 상태 Flag를 읽어 반환한다.
▪ 인수	SPIx - SPI 채널 SPI_I2S_FLAG- 읽어 올 Bit Flag
▪ 반환 값	FlagStatus - Bit Flag의 값 (SET('1') 또는 RESET('0'))
▪stm32f10x_spi.c 참고	

*** 참고사항**

STM32 SPI 관련 실습은 Zigbee를 통한 실습을 진행 한다. 이것은 CD내의 Zigbee관련 프로젝트의 Zigbee/BasicRF/src/hal_ewck.c 내용을 참조하도록 한다.

EWCK-STM32 User Guide

- ATmega128 개발환경 구축 및 I/O 제어 실습 -

EWCK-STM32 개발 보드를 구입해 주신 고객님께 진심으로 감사드립니다. 본 User Guide는 EWCK-STM32를 구입하신 분들께서 EWCK-STM32 보드를 이용하여, 편리하고 쉽게 제품을 개발하실 수 있도록 도와드리기 위해서 제작되었습니다.

본 문서의 모든 저작권은 (주)지비테크에 있습니다. 무단 전제 및 복제를 금합니다.
Copyright© 2010 by GBtec Inc. LTD

▣ Contents

실습 0. ATmega128이란? ····· 97
0.1 ATmega128의 특징 ····· 97
0.2 ATmega128 내부 구조 ····· 98

실습 1. 개발환경 구축하기 ····· 100
1.1 임베디드 시스템 개발 환경 ····· 100
1.2 WinAVR 설치하기 ····· 102
1.3 AVR Studio4 설치하기 ····· 106
1.4 AVR Studio4 실행 및 컴파일하기 ····· 113
1.5 AVR Studio4에서 Program 하기 ····· 116

실습 2. GPIO 제어하기 ····· 122
2.1 GPIO(General Purpose Input/Output) 개요 ····· 122
2.2 GPIO 제어 실습하기 ····· 124

실습 3. Timer/Counter 제어하기 ····· 127
3.1 Timer/Counter 개요 ····· 127
3.2 Timer/Counter 레지스터 ····· 129
3.3 Timer/Counter 제어 실습하기 ····· 131

실습 4. USART 통신 제어하기 ····· 137
4.1 USART 통신 개요 ····· 137
4.2 USART 통신 프레임 형식 및 동작 모드 ····· 139
4.3 USART 레지스터 ····· 140
4.4 USART 통신 제어하기 ····· 143

실습 5. PIR 센서 제어하기 ····· 152
5.1 PIR 센서 이해하기 ····· 152
5.2 PIR 센서 제어 실습하기 ····· 153

실습 6. DC 모터 제어하기 ····· 154
6.1 DC 모터 이해하기 ····· 154
6.2 DC 모터 제어 실습하기 ····· 155

실습 7. Step 모터 제어하기 ····· 160
7.1 PWM(Pulse Width Modulation) 이해하기 ····· 160
7.2 Timer/Counter2 이해하기 ····· 161
7.3 Timer/Counter2 레지스터 ····· 165
7.4 Step 모터 이해하기 ····· 168
7.5 Step 모터 제어 실습 ····· 170

실습 8. GPS 센서 제어하기 ····· 176
8.1 GPS 센서 이해하기 ····· 176
8.2 GPS 센서 제어 실습하기 ····· 177

실습 9. 초음파(Distance) 센서 제어하기 ····· 180
9.1 외부 인터럽트(External Interrupt) 이해하기 ····· 180
9.2 외부 인터럽트 레지스터 ····· 181
9.3 초음파 센서(NT-TS601) 제어 실습하기 ····· 182

실습 10. 조도(CDS)센서, 연기(Smoke)센서 제어하기 ····· 188
10.1 ADC(Analog Digital Converter) 이해하기 ····· 188
10.2 ADC(Analog Digital Converter) 레지스터 ····· 191
10.3 조도(CDS) 센서 제어 실습하기 ····· 193
10.4 연기(Smoke) 센서(GSAT11-P101) 제어 실습하기 ····· 195

실습 11. G-Sensor 제어하기 ····· 199
11.1 TWI(Two-Wire Serial Interface) 이해 ····· 199
11.2 TWI 레지스터 ····· 200
11.3 TWI 제어하기 ····· 203
11.4 G-Sensor(CMA3000) 제어 실습하기 ····· 205

실습 12. 온습도 센서 제어하기 ·····215
12.1 온습도 센서(Humidity & Temperature Sensor) 이해하기 ·····215
12.2 온습도 센서 제어 실습하기 ·····218

실습 13. SPI 제어하기 ·····226
13.1 SPI(Serial Peripheral Interface) 이해하기 ·····226
13.2 SPI 레지스터 ·····227
13.3 SPI 제어 실습하기 ·····228

실습 0. ATmega128이란?

0.1 ATmega128의 특징

ATmega128 마이크로컨트롤러(Micro-controller)는 ATMEL사에서 만든 8비트 RISC(Reduced Instruction Set Computer) 구조로 되어 있는 AVR CPU Core와 프로그램 저장용 플래시, 메모리 및 데이터 저장용 EEPROM으로 구성된 프로세서(Processor)이다.

ATmega128은 저전력 8비트 마이크로컨트롤러로 특징을 정리하면 다음과 같다.

1. 개량된 RISC 아키텍처

- 대부분 1Clock cycle로 실행되는 133개의 강력한 명령어
- 32x8 범용 워킹 레지스터 및 패리패럴 컨트롤 레지스터
- 완전히 정적인 동작
- ATmega128L은 8MHz, ATmega128은 16MHz까지 동작
- 2 cycle로 실행되는 곱셈기

2. Data Memory 및 Program Memory

- Flash Memory 128KB 내장
- Read, Write가 가능한 4KB의 EEPROM
- 4KB의 SRAM 내장, 외부 Memory를 64KB까지 확장 가능
- Lock 기능, ISP(In-System Programming) 가능

3. JTAG Interface

- Flash Memory, EEPROM, Lock bit 쓰기 가능 및 Debugging 기능 지원

4. 주변 장치

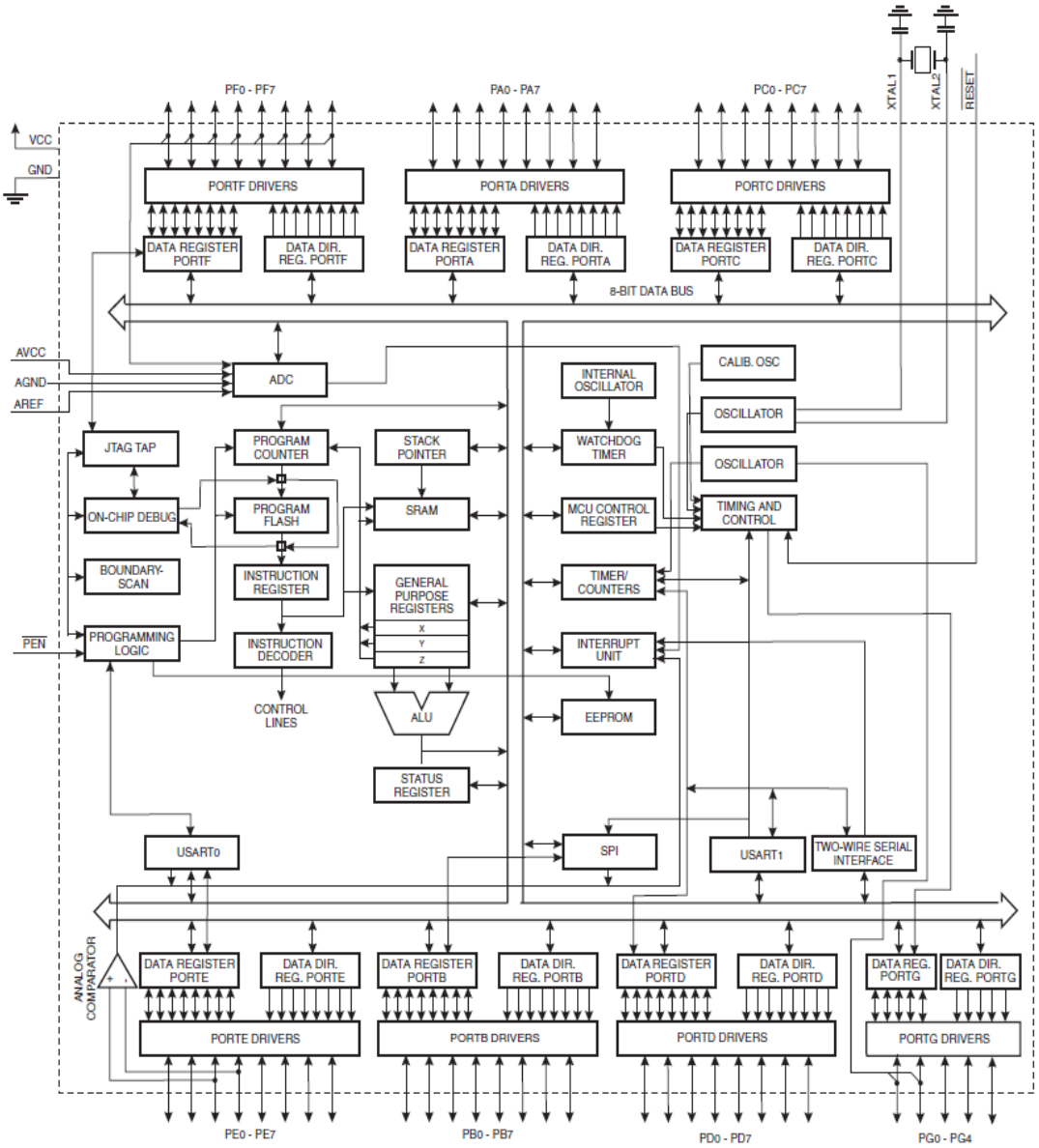
- Prescaler 및 비교 모드 기능이 있는 2개의 8bit Timer/Counter
- Prescaler 및 비교 모드, 캡처 모드 기능이 있는 2개의 16bit Timer/Counter
- RTC(Real Time Counter), Watch dog Timer
- 8bit PWM 채널 2개 및 2~16bit 분해 가능한 PWM 채널 6개, 비교기 출력 가능
- 10bit ADC 8 채널(단독(8채널) 혹은 차동 채널(7채널)) 및 아날로그 비교기
- 마스터/슬레이브 SPI, I2C, USART 2개

5. 특별한 기능

- Power-on Reset, Brown-out 검출
- RC Oscillator 내장 및 Clock 주파수 선택 가능
- Sleep Mode : Idle, ADC noise 감소, Power 절약, Power-down, Standby, Extended Standby
- ATmega103과 호환성 모드 선택 가능 및 전체 Pull-up 동작 불가능 모드 선택 가능
- ATmega128L은 2,7~5.5V, 0~8MHz, ATmega128은 4.5~5.5V, 0~16MHz 동작 가능

0.2 ATmega128 내부 구조

ATmega128의 내부 구조는 다음 그림과 같이 구성되어 있으며, 각각의 기능들을 자세히 설명하면 다음과 같다.



1. Program Memory

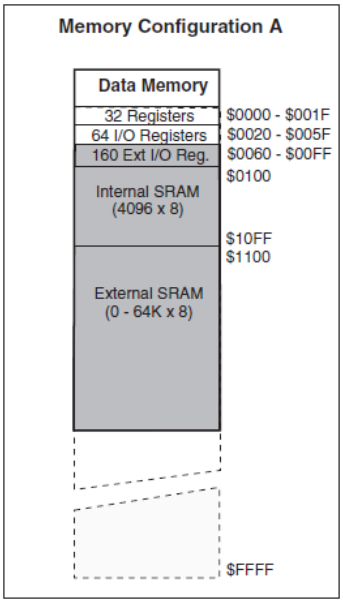
Atmega128은 프로그램 영역으로 128KB의 Flash Memory를 내장하고 있고, 모든 명령은 16bit, 혹은 32bit로 되어 있기 때문에, Flash Memory의 크기는 64x16(64K 워드 혹은 128KB)이 된다.

- ATmega128의 PC(Program Counter)는 16bit이므로, 프로그램 영역은 64K 워드의 주소를 지정할 수 있다.
- 부트 프로그램 영역과 응용 프로그램 영역으로 나누어진다.
- SPI를 이용한 ISP 방법 혹은 JTAC 또는 병렬로 써넣을 수 있다.
- Flash Memory는 10,000번 써넣기, 지우기를 할 수 있다.

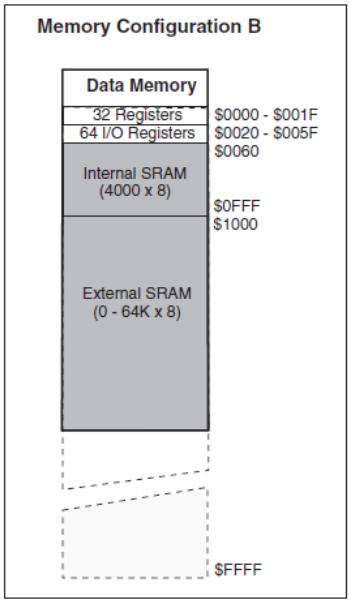
2, Data Memory

ATmega128은 다음 표와 같이 일반 모드(Normal mode), ATmega103 호환 모드(ATmega103 Compatibility mode)에 따라 다른 Memory 구조를 지원한다. 그 아래의 그림은 각 Data Memory 구조와 레지스터 파일 구조이다.

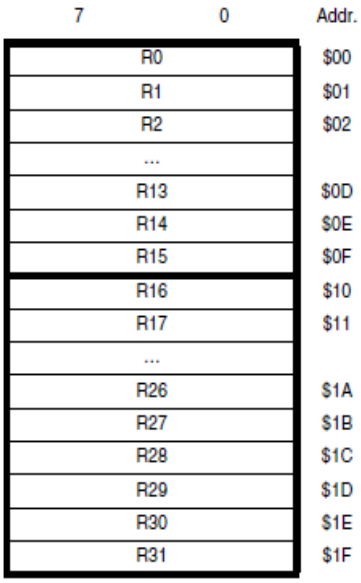
Configuration	Internal SRAM Data Memory	External SRAM Data Memory
Normal mode	4096	up to 64K
ATmega103 Compatibility mode	4000	up to 64K



[Normal mode]



[ATmega103 호환 모드]



[레지스터 파일]

- 확장된 I/O 영역은 ATmega103 호환 모드에서는 존재하지 않는다.
- 외부에 SRAM을 사용할 경우는 MCUCR 레지스터의 SRE 비트를 '1'로 설정하여 외부 SRAM을 액세스 할 수 있으며, Read/Write 제어 신호로 PG0과 PG1 Pin을 사용하여 내부 Data Memory처럼 액세스 할 수 있다.
- 레지스터 파일은 위의 그림과 같으며 실습에서의 모든 예제는 C언어를 사용하므로 레지스터 파일의 레지스터 기능을 자세히 알 필요는 없다.
- I/O 레지스터는 각 장에서 필요에 따라 살펴볼 것이며, 자세한 내용은 데이터 시트를 참조한다.

3. EEPROM Data Memory

전원이 켜져 있는 동안, 즉 프로세서가 동작하는 동안 내부 프로그램 코드에 의해 발생되는 데이터를 일시적으로(전원이 켜져 있는 동안) 저장하기 위해 필요한 SRAM과는 달리 EEPROM은 전원이 꺼지더라도 지속적으로 그 값을 유지해야 할 필요가 있는 별도의 데이터를 저장하기 위해 사용되는 Memory이다.

총 4KB의 용량을 가지는 ATmega128 내부 EEPROM 영역을 접근하기 위해서는 ATmega128 내부의 별도 레지스터들(EEAR, EEDR, EECR)을 제어함으로써 가능하다.

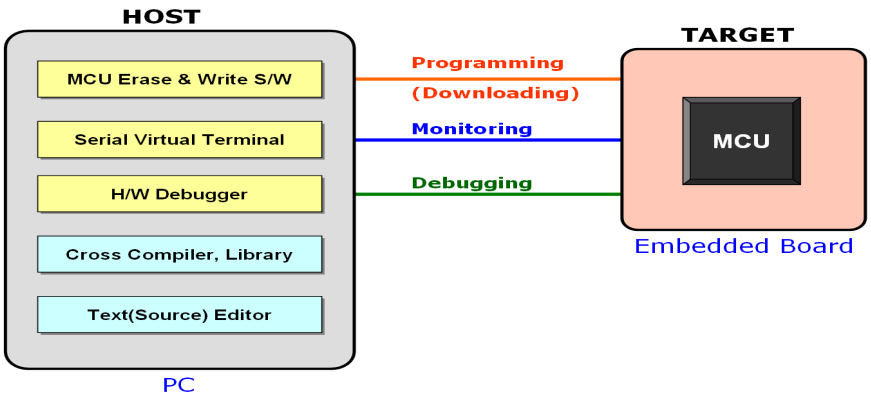
실습 1. 개발환경 구축하기

1.1 임베디드 시스템 개발 환경

실습 보드 개발 환경을 구축하기 전에 일반적으로 임베디드 시스템을 개발하기 위해서 어떻게 환경을 구축하여 사용하고 있는지부터 살펴보자.

임베디드 소프트웨어 개발자가 아닌 일반 프로그래머들은 PC 환경에서 개발하며, 그렇게 개발한 프로그램 역시 PC 환경에서 실행하기 때문에 모니터와 키보드, 마우스와 같은 표준 입출력 하드웨어들을 모두 구비하고 있는 PC 환경 하에서 개발할 때에는 이러한 개발 환경 때문에 고민하는 일은 보통 하지 않는다.

그러나 임베디드 소프트웨어를 처음 접하는 개발자들이 가장 먼저 닥치는 난관이 바로 개발 환경 구축에 관련된 것이다. 임베디드 소프트웨어란 자체가 PC와는 달리 표준 입출력 하드웨어가 전혀 없고, 덩그러니 보드(Board)만 있는 플랫폼에서 개발해야 하기 때문에 익숙하지 않은 개발자들에겐 난감한 문제가 아닐 수 없다. 따라서 이에 대한 내용을 먼저 이해하고 넘어가도록 하자.



먼저, 우리가 프로그램 소스 코드를 개발하는 PC를 일반적으로 호스트(Host)라고 하며, 개발의 목적이 되는 임베디드 보드를 일반적으로 타겟 보드(Target Board)라고 말한다. 이후 용어의 통일성을 위하여 호스트와 타겟 보드라는 용어를 사용하여 설명하도록 한다.

임베디드 시스템 개발 환경을 구축한다는 의미는 바로 위의 그림과 같이 호스트 환경을 구축한다는 의미이다. 이러한 호스트 환경은 크게 하드웨어 환경과 소프트웨어 환경으로 나눌 수 있는데, 먼저 하드웨어 환경은 호스트와 타겟 보드 간 데이터를 주고받기 위해 케이블로 연결한다는 의미이며, 소프트웨어 환경은 이렇게 연결된 케이블에 데이터를 주고받기 위해 호스트 상에 해당 소프트웨어들을 설치하고 설정하는 것을 의미한다.

이러한 하드웨어, 소프트웨어 환경은 호스트와 타겟 보드 간에 주고받는 데이터에 따라 위의 그림처럼 크게 세 부분으로 나뉘고, 추가로 호스트에 Cross compiler와 Library가 있는데 이를 세부적으로 살펴보면 다음과 같다.

1. Programming (Downloading)

타겟 보드를 구동하기 위한 프로그램 코드들을 타겟 보드에 내장된 메모리나 혹은 해당 프로세서 내부에 내장된 프로그램 메모리에 저장(Programming 혹은 Fusing)하기 위한 환경을 말하며 이를 위해 별도의 케이블과 별도의 전용 소프트웨어가 필요하다.

일반 사용자들이 보아왔던 32비트의 고 사양 프로세서의 경우엔 프로세서 내부에 프로그램 코드용 메모리가 따로 없으므로 이 경우에는 보통 타겟 보드에 장착된 외부 플래시 메모리로 해당 프로그램 코드를 저장하며, ATmega128의 경우에는 프로세서 내부에 프로그램 메모리가 따로 있으므로 여기에 저장할 수 있다.

우리가 사용할 AVR Studio4는 Program 기능까지 제공하고 있으므로, 우리는 이것을 이용할 것이다.

2. Monitoring

타겟 보드는 호스트와 달리 모니터와 키보드, 마우스 등의 표준 입출력 하드웨어가 없는 경우가 대부분이기 때문에 타겟 보드에 명령을 입력하거나 타겟 보드의 상태를 확인하기 위해 시리얼(Serial) 연결을 통한 터미널 응용 프로그램(예: 하이퍼터미널)이 사용된다. 이를 위해서 별도의 케이블 연결과 소프트웨어 설치 및 설정을 필요로 한다.

하이퍼터미널은 [시작]-> [모든 프로그램]-> [보조 프로그램]-> [통신]-> [하이퍼터미널]에서 찾을 수 있다.

3. Debugging

앞서 언급하였듯이 타겟 보드는 별도의 입출력 하드웨어가 없기 때문에 내부 상태를 호스트의 가상 터미널을 통해 소프트웨어적으로 확인해 보거나, 혹은 하드웨어 전용 장비를 통해 하드웨어적으로 확인해 볼 수 있다. 이러한 하드웨어 디버깅을 위해서 별도의 케이블과 에뮬레이터와 같은 전용 하드웨어 장비, 또 그에 따르는 전용 소프트웨어를 구축하여 사용한다.

AVR 전용 에뮬레이터 등을 사용하는 방법이 있지만, 사용법이 어렵고 장비도 상대적으로 비싸기 때문에 여기서는 간단한 디버깅 방법으로 UART 통신 모듈을 소스 코드에 넣어 디버깅하도록 한다.

4. Tool Chain - Cross compiler & Library

Programming과 Monitoring, Debugging 환경은 이미 타겟 보드용 프로그램 코드가 개발된 상태에서 필요한 환경이라면, Tool Chain은 그런 타겟 보드용 프로그램 코드를 개발하기 위해 필요한 소프트웨어 환경을 말한다.

임베디드 소프트웨어란 결국 타겟 보드를 구동하기 위해 필요한 소프트웨어라고 할 수 있다. 일반적으로 PC 상에서 실행되는 소프트웨어를 PC 상에서 개발하는 것과 달리, 타겟 보드는 타겟 보드 상에서 개발할 수 없기 때문에 타겟 보드용 소프트웨어는 호스트 상에서 개발할 수밖에 없다.

그러기 위해 필요한 컴파일러를 Cross Compiler라 하며, 이때 필요한 라이브러리(Library)와 관련 유틸리티(binutil)들을 일컬어 Tool Chain이라 한다. 이는 임베디드 개발 환경 구축 시 가장 먼저 구축해야 하는 부분으로 알려져 있다.

ATmega128은 전용 무료 Cross compiler인 AVR-GCC가 공개되어 있지만, 처음 다루는 사용자들에게는 명령어 라인의 입력 방법이나 Makefile의 구성 방법과 같은 곳에서 어려움을 느낄 수 있다. 하지만 WinAVR, AVR Studio와 같은 무료 소프트웨어를 이용하여 쉽고 간단하게 컴파일 할 수 있는 환경을 갖출 수 있으므로, 우리는 이것을 사용하여 실습할 것이다.

ATmega128 보드 실습을 위해 우리가 설치해야 할 소프트웨어는 다음과 같다.

- WinAVR (Tool Chain) : http://sourceforge.net/projects/winavr/
- AVR Studio (Tool Chain & Programming tool) : http://atmel.com/

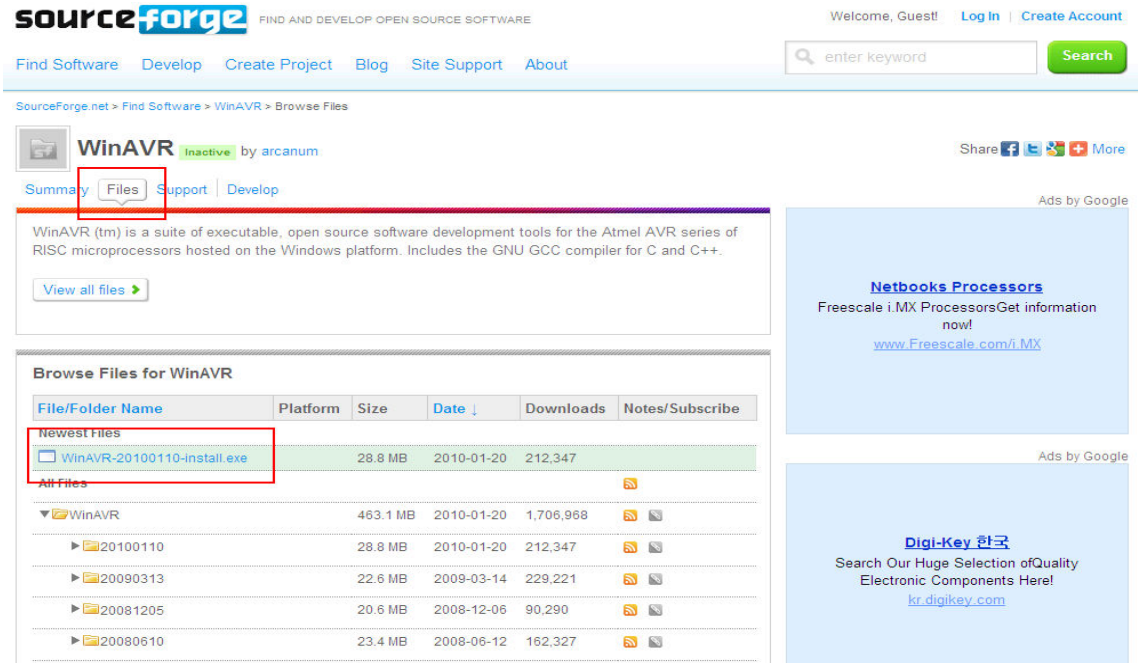
1.2 WinAVR 설치하기

AVR은 8비트 프로세서들 중 유일하게 강력한 공개용 컴파일러인 GCC가 지원되는 프로세서이다. AVR을 지원하기 위한 GCC를 AVR-GCC라 하며, IAR이나 CodeVision 등 많은 컴파일러가 존재한다. 우리는 무료로 배포되는 WinAVR과 AVR Studio4를 이용할 것이다.

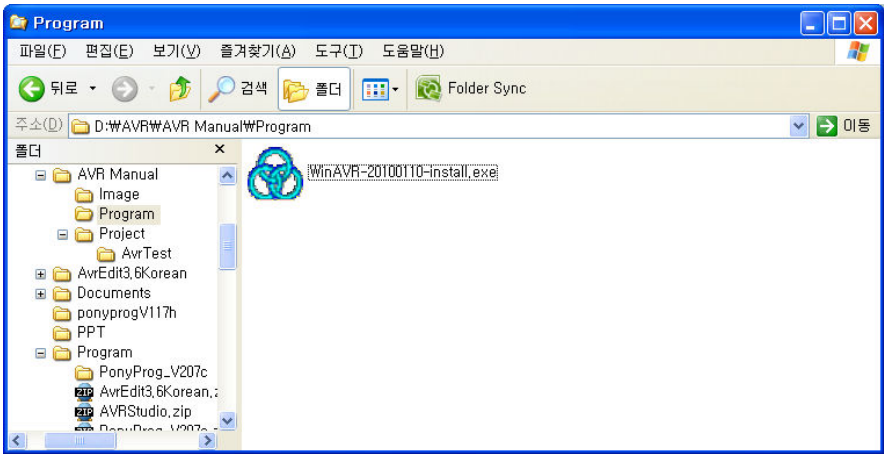
아래 사이트에서 최신 버전의 WinAVR 설치 파일을 다운로드 할 수 있다.

http://sourceforge.net/projects/winavr/

다음 화면과 같이 Files 탭을 선택한 후 'Newest Files' 하단의 실행 파일을 선택하거나, 아래의 폴더에서 원하는 릴리즈 날짜를 선택하여 파일을 다운로드 받을 수 있다.



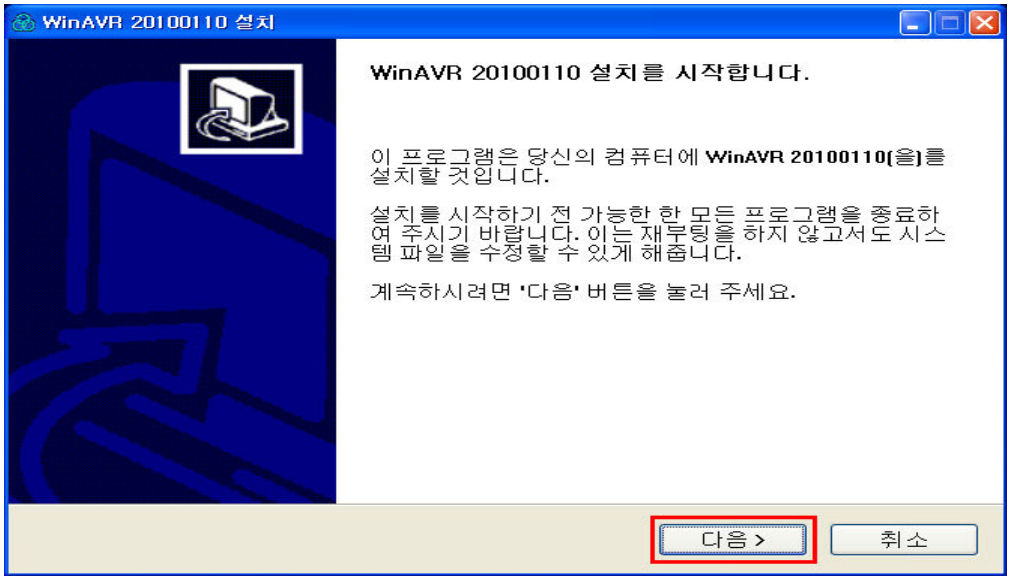
저장된 설치 파일을 실행한다.



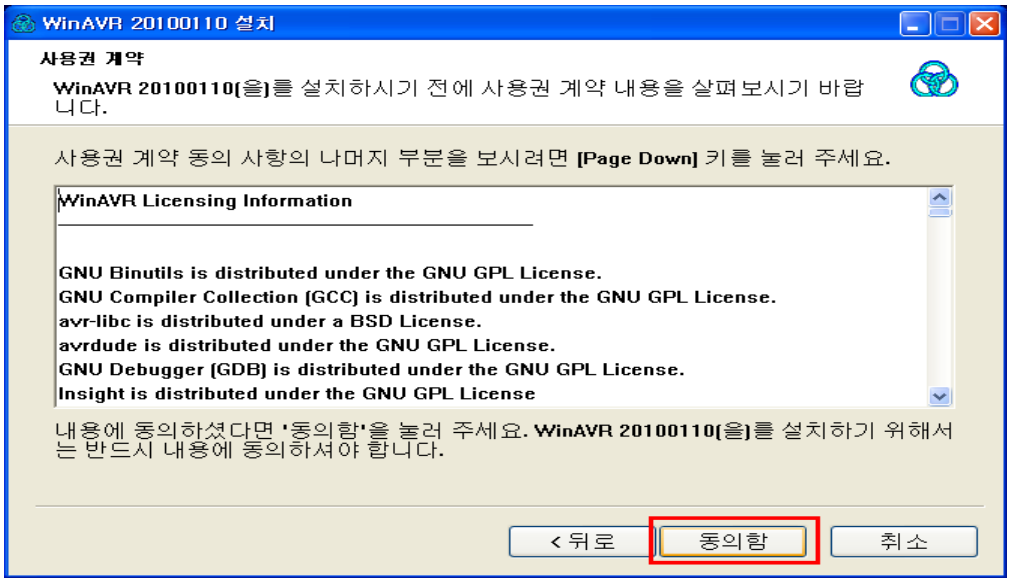
다음 설치 단계에 따라 설치를 진행한다.

처음 나타나는 화면은 사용 언어를 선택하는 화면으로 'Korean'(한국어)을 선택한 후 'OK' 버튼을 클릭한다.

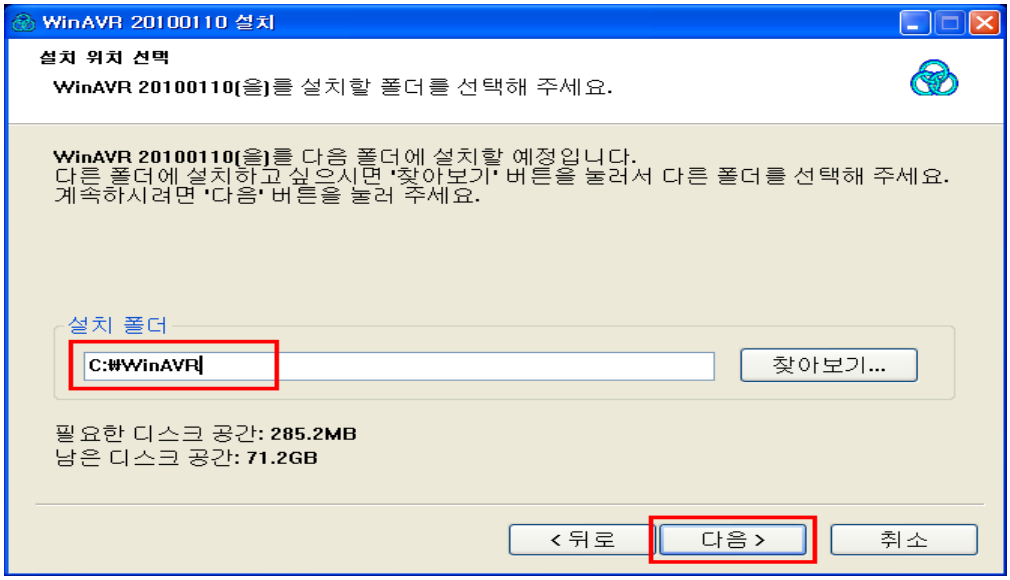
설치 초기 화면이다. '다음' 버튼을 클릭한다.



라이선스 관련 화면이며 WinAVR에 구성되는 각 모듈들의 사용권에 관한 내용들을 표시하고 있다. 사용 내용에 동의해야 설치가 가능하므로 '동의함' 버튼을 클릭한다.

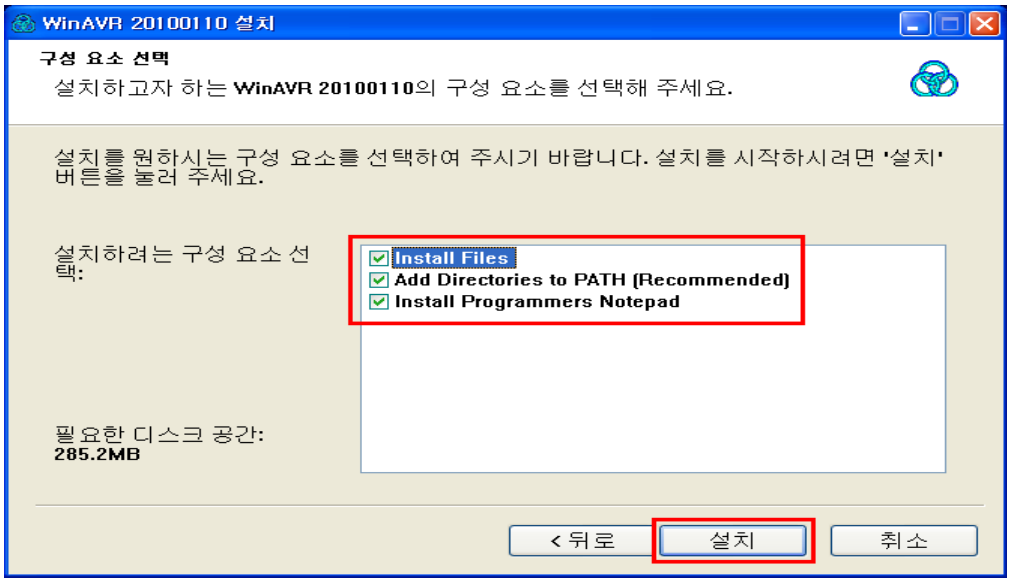


설치 폴더를 선택한다. 기본적으로 WinAVR 뒤에 버전의 날짜가 함께 입력되어 있지만, Makefile의 경로에 영향을 끼칠 수 있으므로 설치 폴더는 반드시 C:\WinAVR로 지정한다.

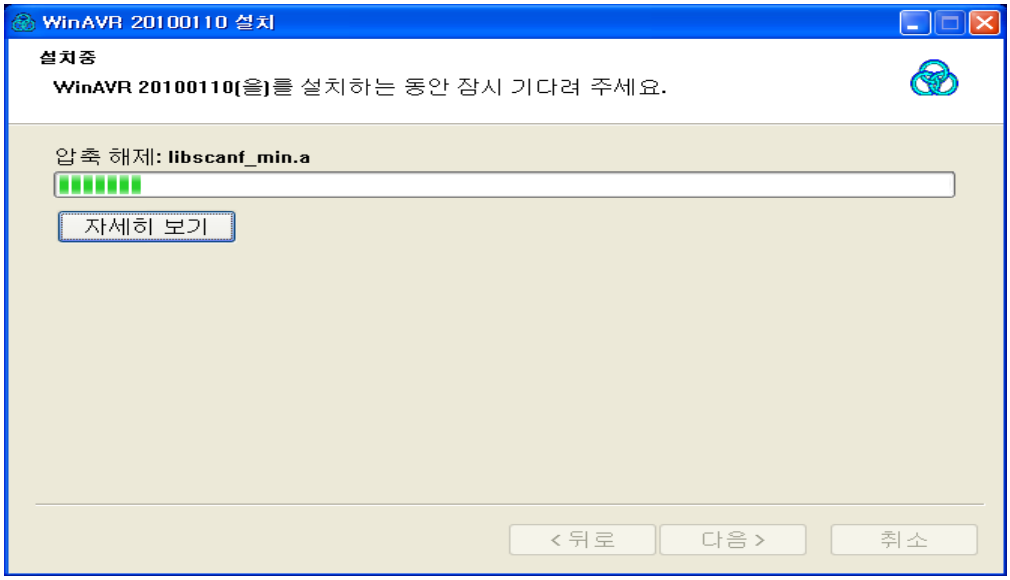


구성 요소를 선택하는 화면이다. 여기에서 'Install Files'는 기본적인 설치 파일들을 의미하고, 'Add Directories to PATH (Recommended)'는 WinAVR 설치 파일들의 경로를 Windows 환경변수에 지정하는 것을 의미한다. 'Install Programmers Notepad'는 C언어 소스 편집기인 Programmers Notepad를 설치하는 것을 의미한다.

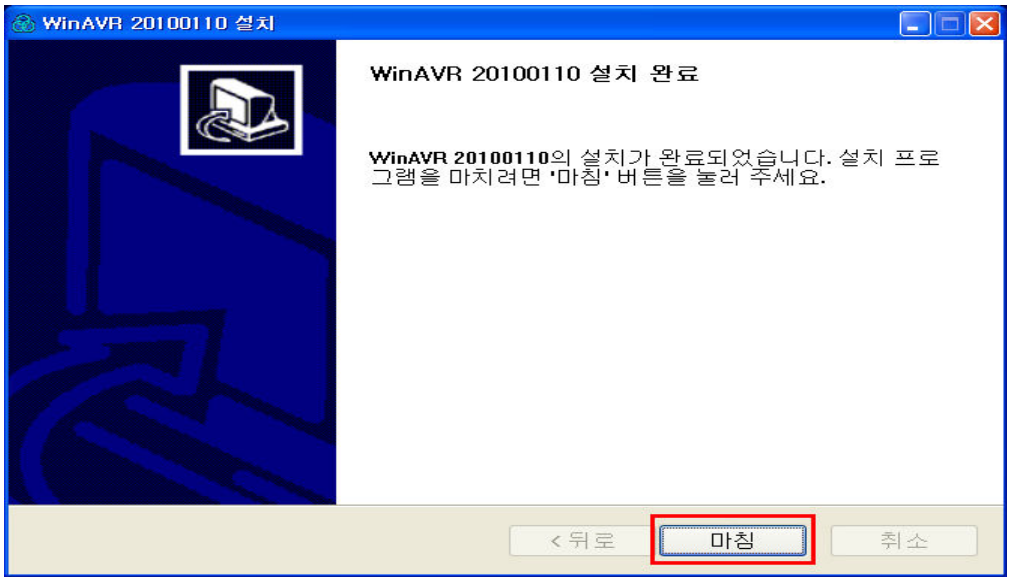
여기서는 기본적으로 모두 체크한 후 '설치' 버튼을 클릭하면 설치를 시작한다.



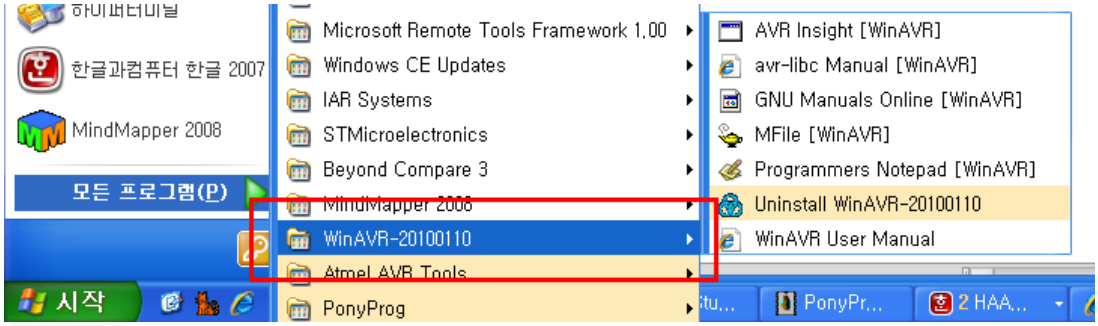
설치 진행 중인 화면이다.



설치가 완료되면 '마침' 버튼을 클릭하여 설치를 종료한다.



[시작]->[모든 프로그램]에서 설치된 프로그램을 확인할 수 있다.

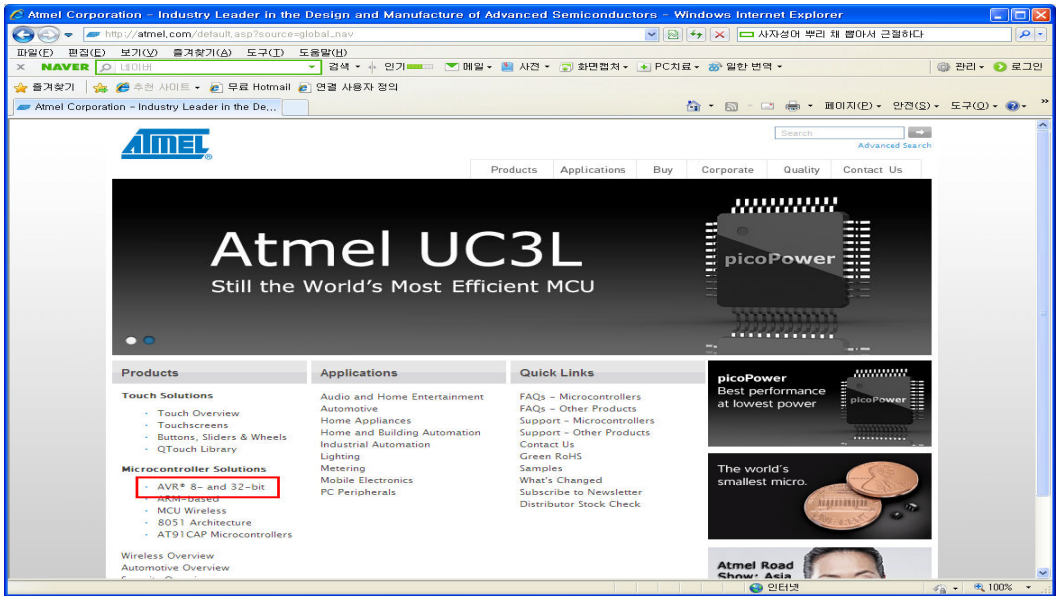


1.3 AVR Studio4 설치하기

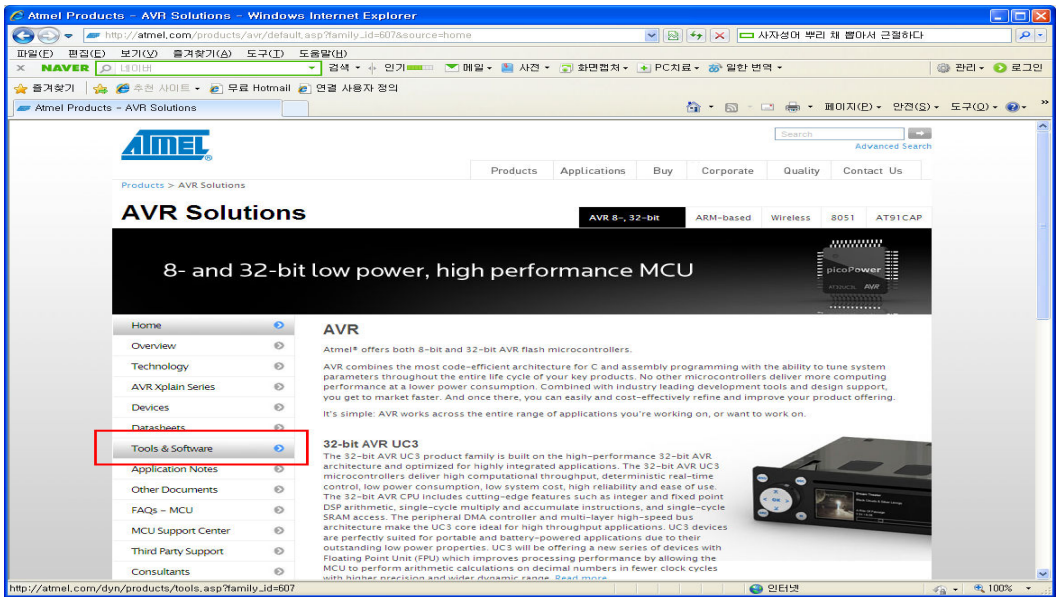
WinAVR이 라이브러리와 컴파일러를 제공한다면, AVR Studio4는 사용자가 8비트 AVR 어플리케이션 개발을 쉽게 할 수 있도록 하기 위한 통합 개발 환경을 제공한다. 설치 프로그램은 아래의 사이트에서 다운로드 할 수 있다. 단, AVR Studio가 설치되는 과정에서 자동으로 WinAVR 폴더를 참조하므로, WinAVR이 먼저 설치되어 있어야 컴파일이 정상적으로 실행된다.

http://atmel.com/

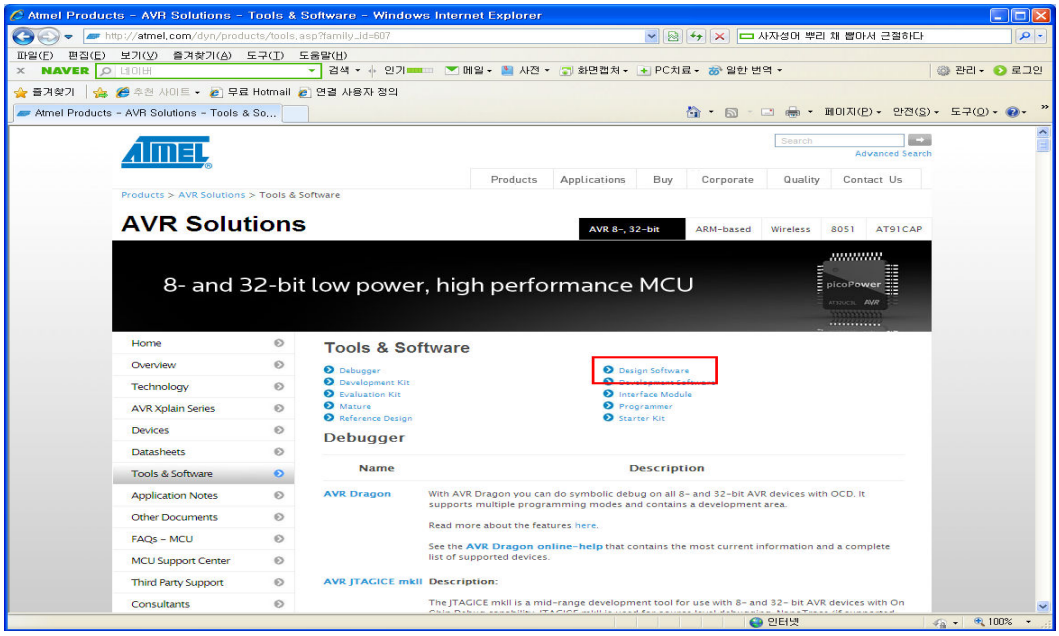
"AVR* 8- and 32-bit" 메뉴를 선택한다.



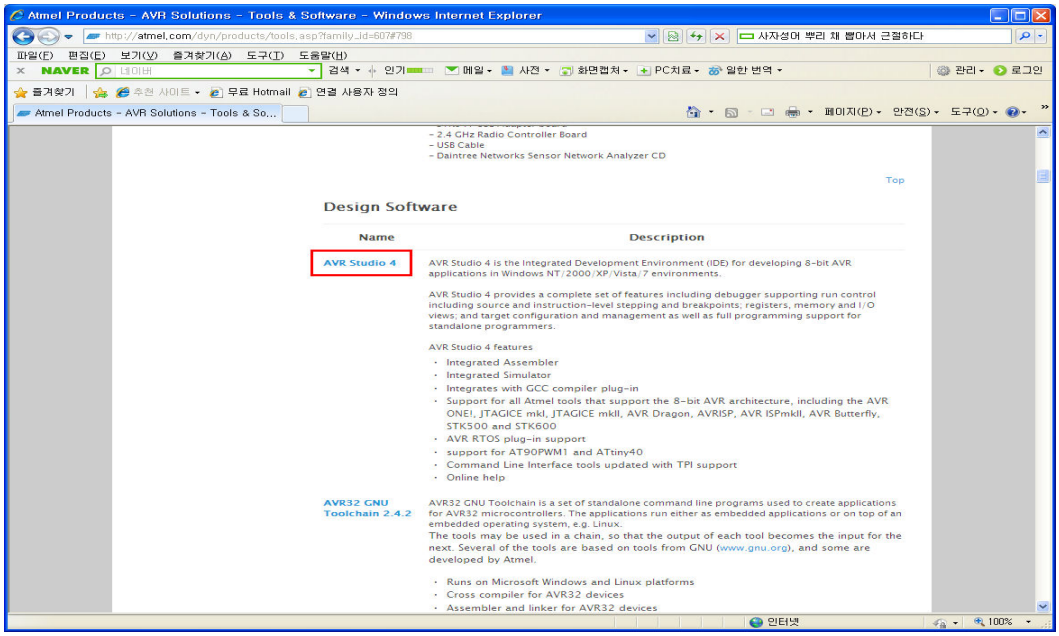
"Tools & Software" 메뉴를 선택한다.



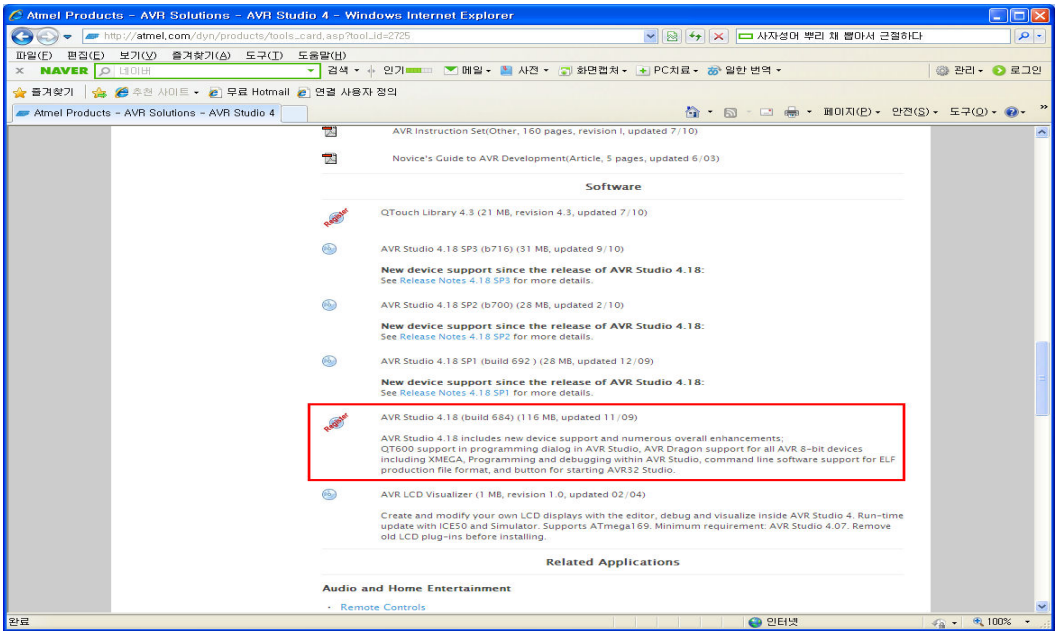
"Design Software" 메뉴를 선택한다.



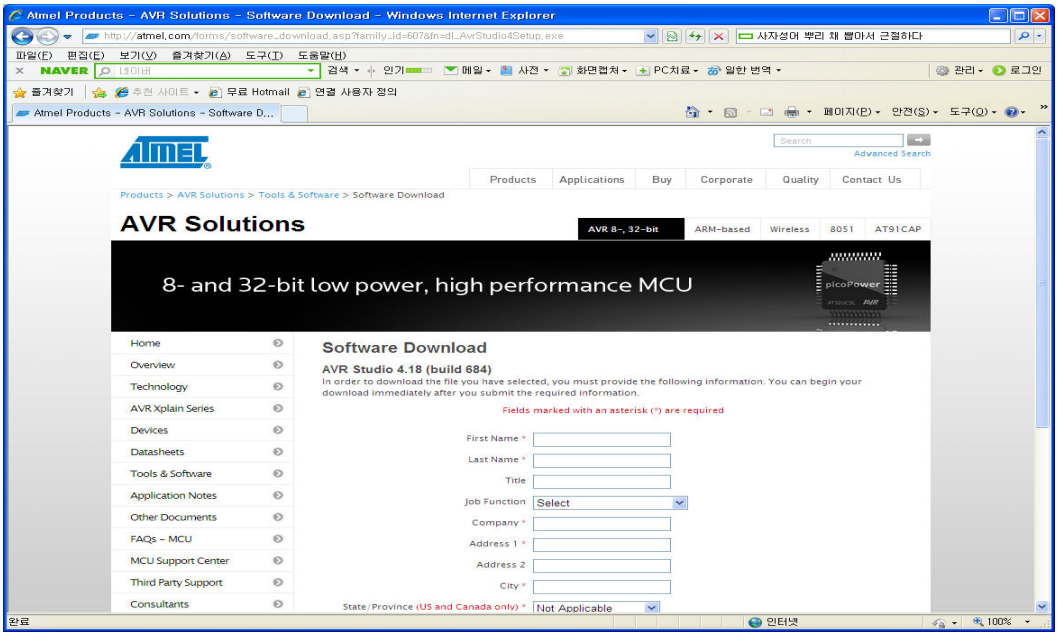
"AVR Studio 4"를 선택한다.



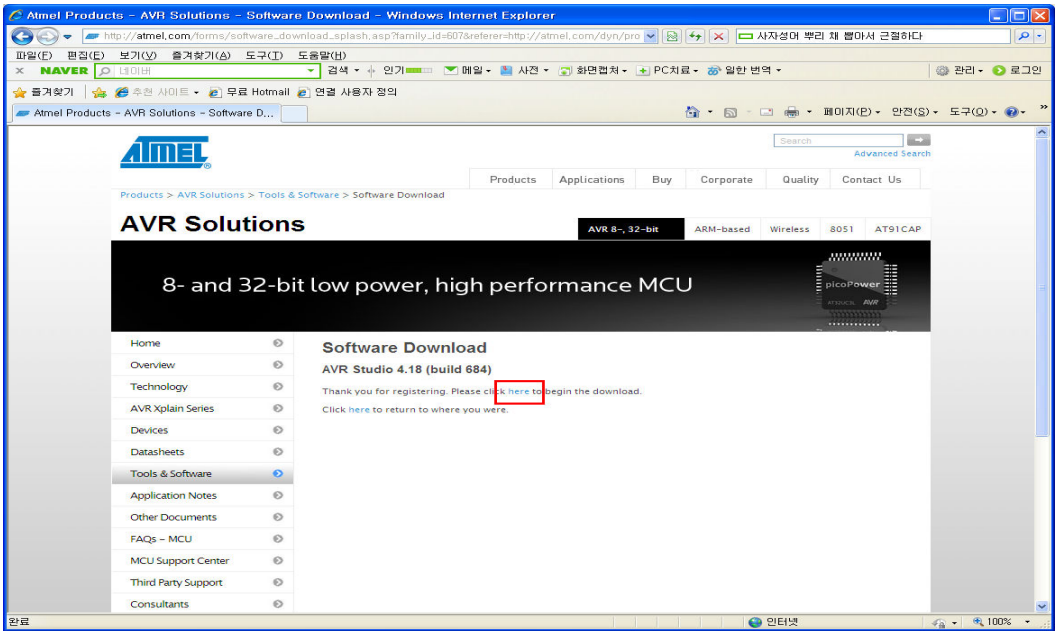
"AVR Studio 4.18" 버전을 선택한다. SP1~3이 있는데, 여기서는 따로 설치하지 않아도 상관없다.



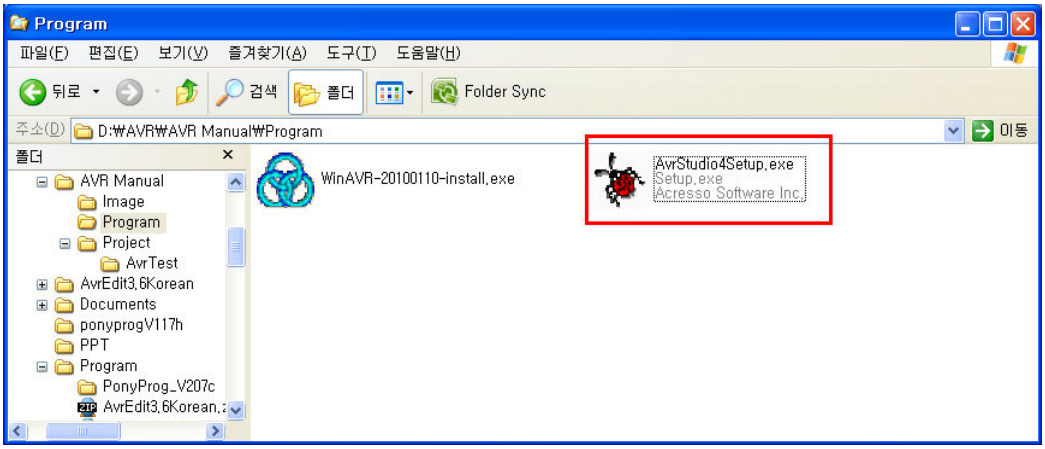
기본적인 정보를 입력 후 'Click to Download' 버튼을 클릭한다.



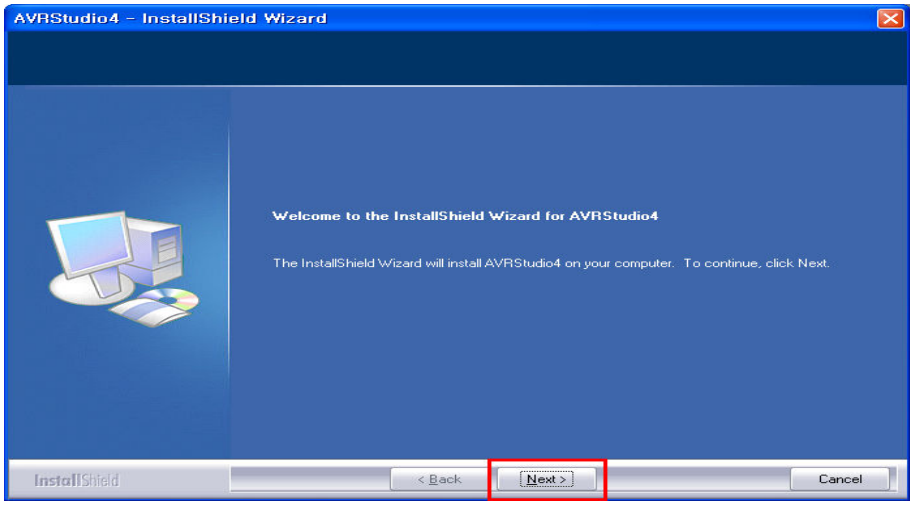
'here' 를 클릭하면 설치 파일을 다운로드 할 수 있다. 아랫줄의 'here' 를 선택하면 이전의 정보 입력 화면으로 돌아간다.



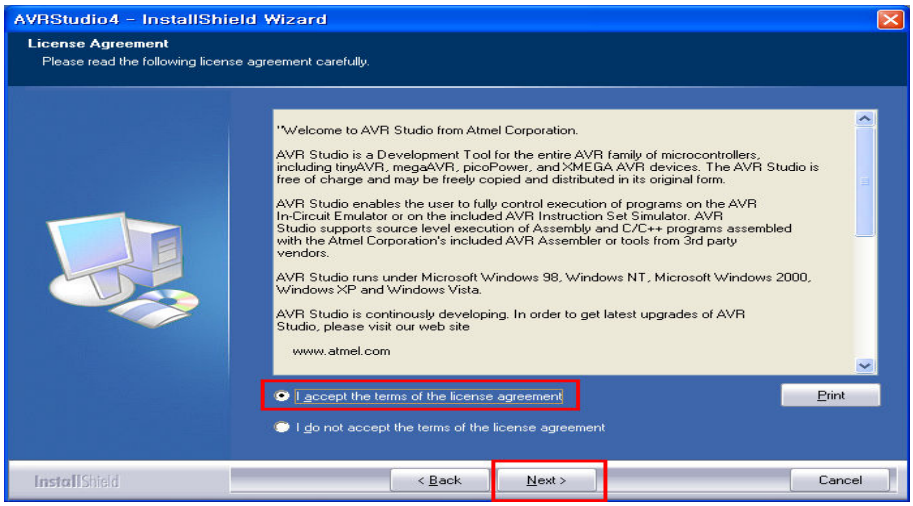
설치 파일을 실행하여 설치를 시작한다.



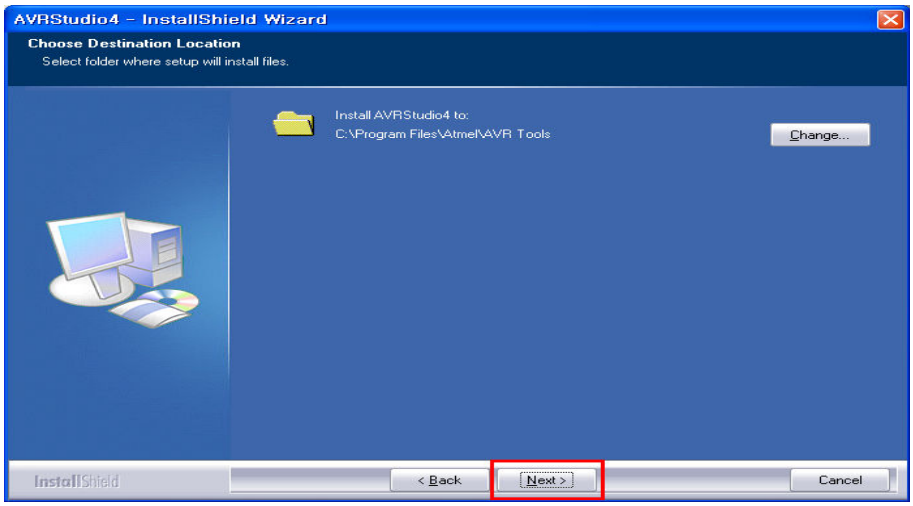
설치 초기 화면이 나타나면 'Next' 버튼을 클릭하여 설치를 시작한다.



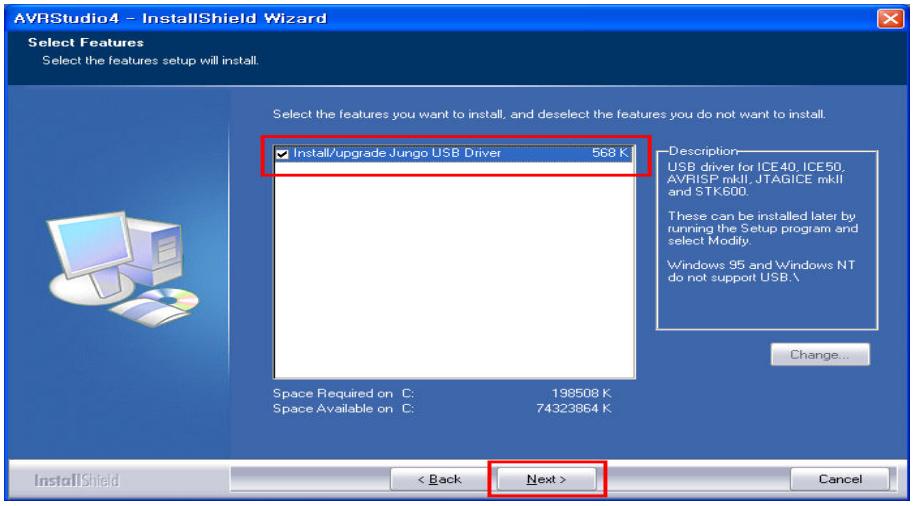
라이선스 동의 화면이다. 'I accept the terms of the license agreement'를 선택하고 'Next' 버튼을 클릭한다.



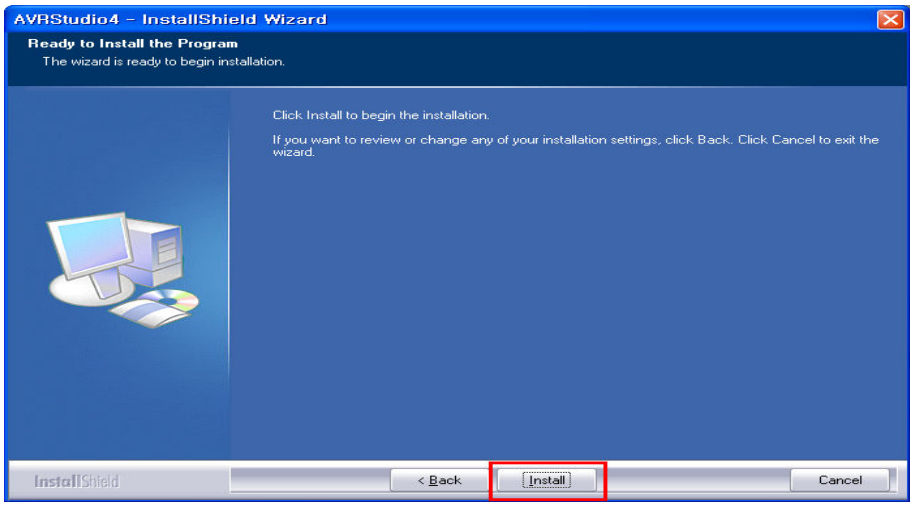
AVR Studio4가 설치될 폴더를 지정한 후 'Next' 버튼을 클릭한다.



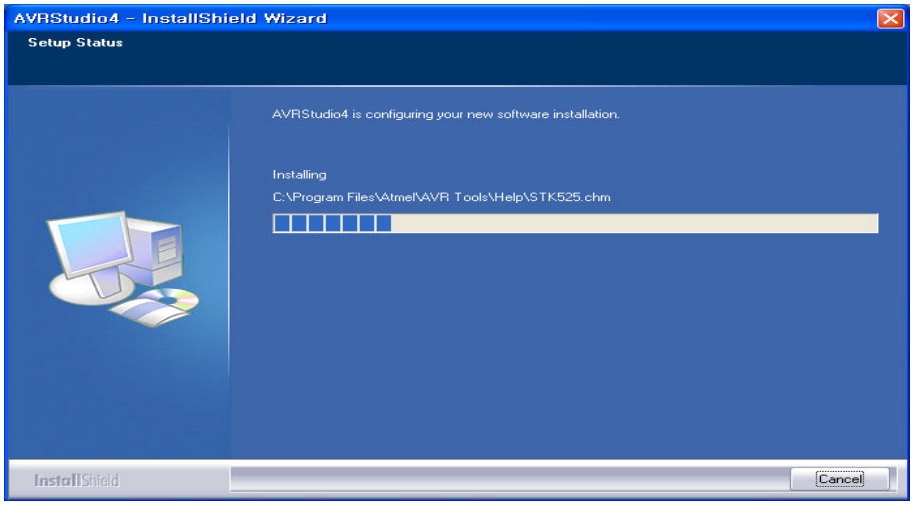
USB 드라이버 설치 관련 화면이다. ‘Install/upgrade Jungo USB Driver’ 에 체크한 후 ‘Next>’ 를 선택한다.



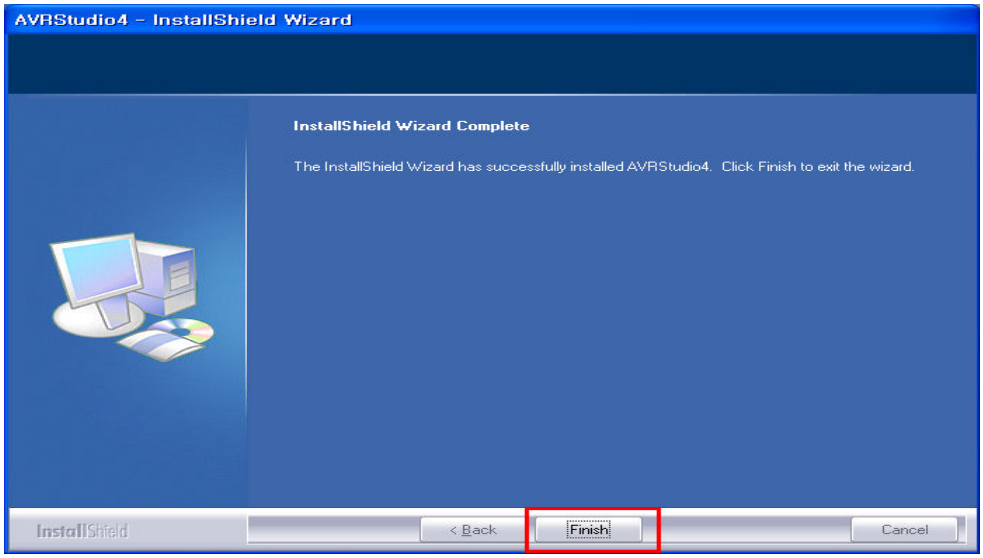
‘Install’ 버튼을 클릭하면 설치를 시작한다.



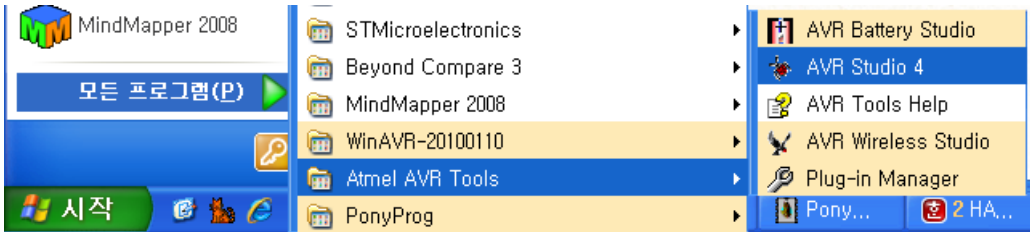
설치가 진행 중인 화면이다. 잠시 기다리면 설치가 완료된다.



설치가 완료되면 'Finish' 버튼을 클릭하여 설치를 종료한다.



[시작]-> [모든 프로그램]에서 설치된 프로그램을 확인한다.

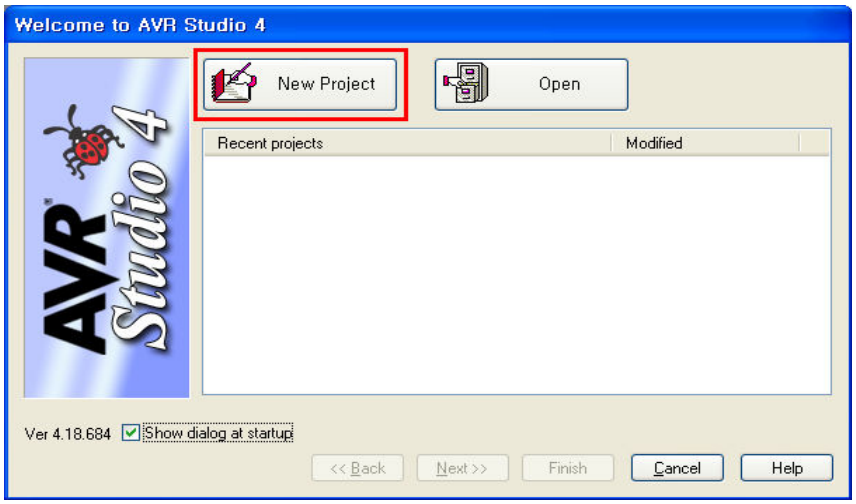


1.4 AVR Studio4 실행 및 컴파일하기

설치된 AVR Studio를 실행하여 간단한 코드를 작성하고, 컴파일러가 제대로 동작하는지 확인해 보도록 하자.

AVR Studio4를 실행하면 다음과 같이 프로젝트를 생성할 것인지를 묻는 화면이 나타난다. 이전에 작업했던 프로젝트가 있을 경우 'Recent projects' 목록에서 선택할 수 있으며, 'Open'을 선택할 경우 직접 프로젝트를 찾아 실행할 수 있다.

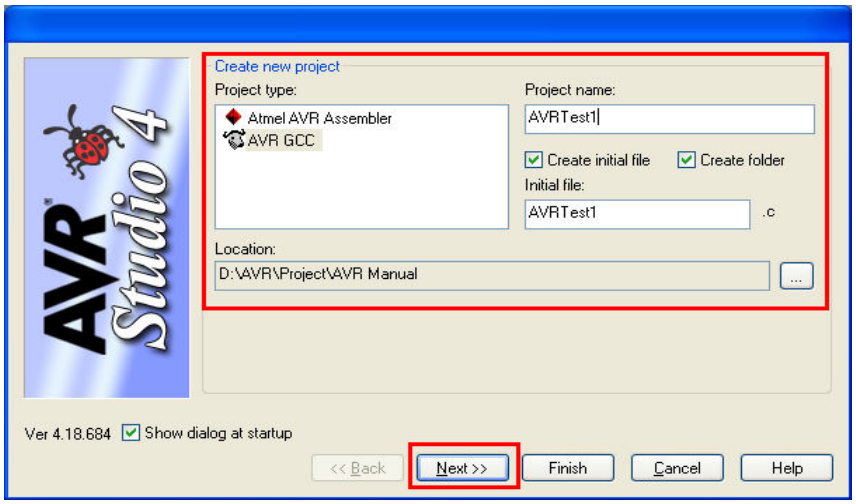
지금은 새로운 프로젝트를 생성할 것이므로 'New Project'를 선택한다. 또는 메뉴의 [Project]-> [New project]를 선택하여 새로운 프로젝트를 생성할 수 있다.



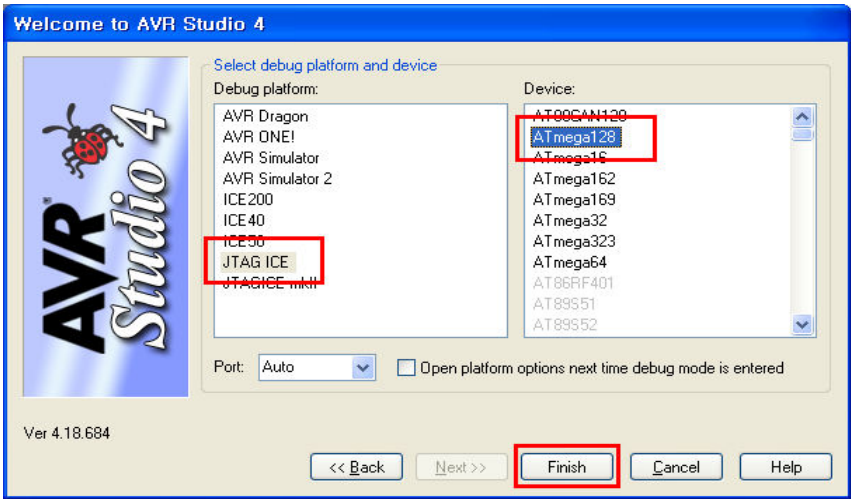
Project type을 선택한다. 'Atmel AVR Assembler'는 컴파일 시 AVR Studio4에 내장된 어셈블러를 사용하며, 'AVR GCC'는 사용자가 지정한 외부 컴파일러를 사용하여 컴파일을 실행한다. 여기에서는 WinAVR의 컴파일러와 링커를 사용할 것이므로 'AVR GCC'를 선택한다.

'Project name'에 프로젝트명을 입력 시 'Create initial file'에 체크가 되어 있으면 자동으로 Initial file에 파일명이 입력되고 해당 파일이 생성된다. 'Create folder'에 체크되어 있을 경우 프로젝트 이름의 폴더가 생성되고 하위에 프로젝트 파일이 생성된다. 두 군데 모두 체크하는 것을 권장한다.

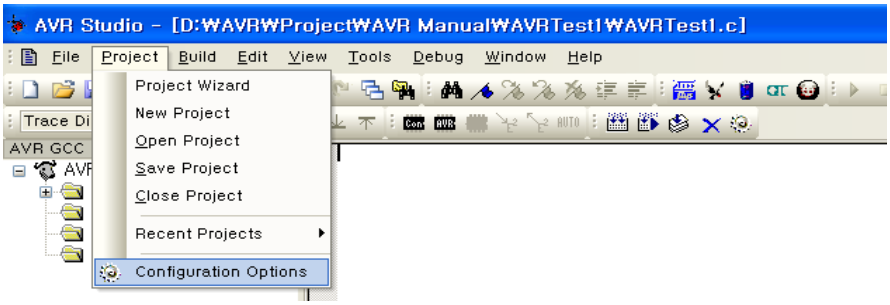
'Location'에는 프로젝트가 생성될 경로를 지정한다. 이 때 경로에 한글이 포함될 경우, 컴파일 시 'make: Makefile: No such file or directory'라는 오류가 발생하므로 경로 지정에 유의하도록 한다.

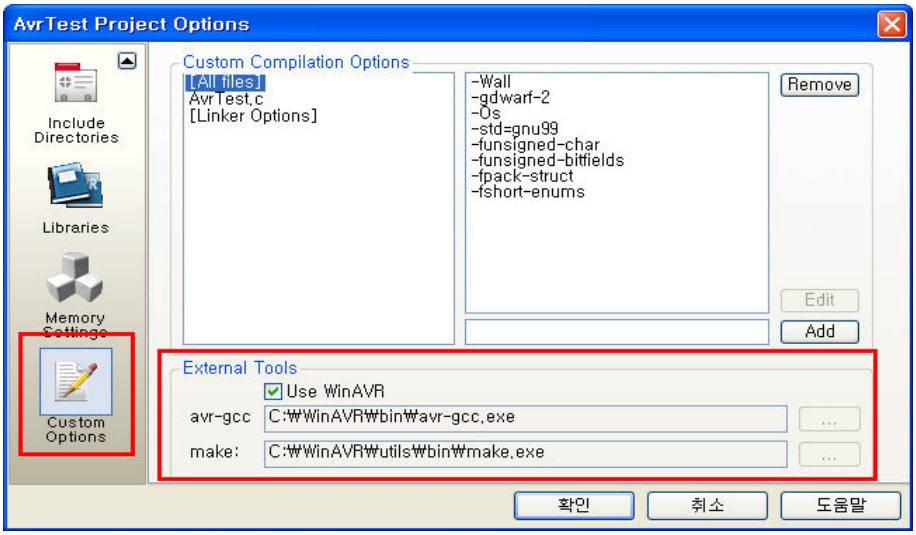


Debug platform과 Device를 선택한 후, 'Finish' 버튼을 클릭한다.



[Project]-> [Configuration Options] 메뉴를 선택하여 'Custom Options-> External Tools' 가 WinAVR로 설정되어 있는지 확인한다. 정상적으로 설치되었다면 아래 화면과 동일하게 설정되어 있을 것이다.



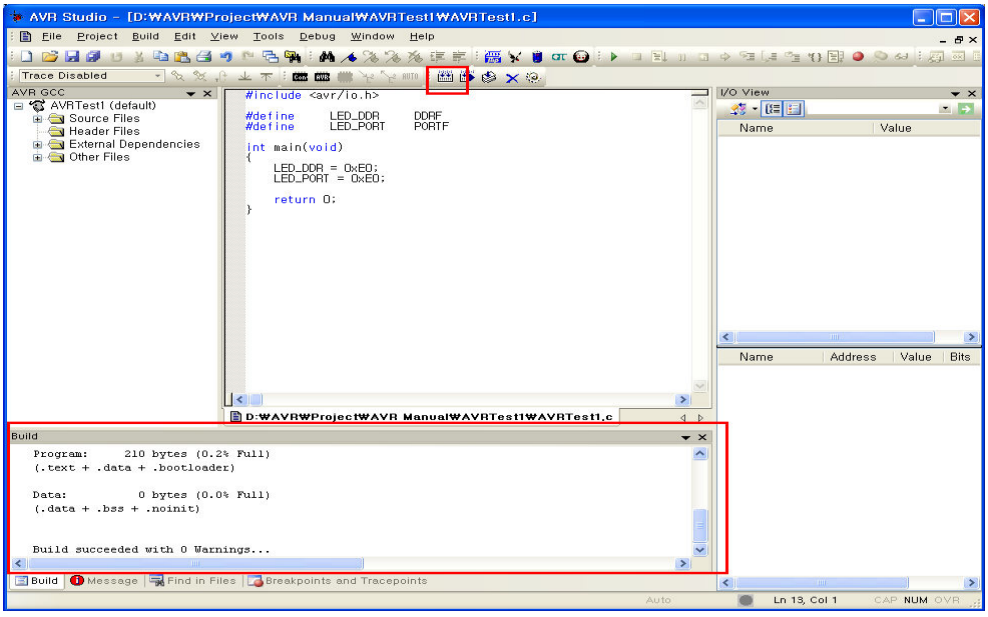


컴파일러가 정상적으로 동작되는지 확인하기 위해 다음의 소스를 입력한다. 아래 소스는 단순히 LED를 켜는 동작만을 하는 소스이다.

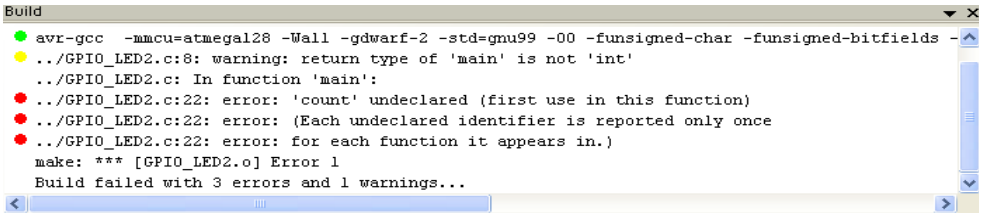
▣ Source Code (Project Name - AVRTest1)

```
#include <avr/io.h>

#define   LED_DDR DDRF
#define   LED_PORT        PORTF

int main(void)
{
        LED_DDR = 0x64;
        LED_PORT = 0x64;

        return 0;
}
```

F7 또는 [Build]-> [Build]를 선택하여 빌드 한다. 다음 화면과 같이 'Build succeeded with 0 Warnings...' 라고 나타나면 컴파일이 정상적으로 된 것이다.



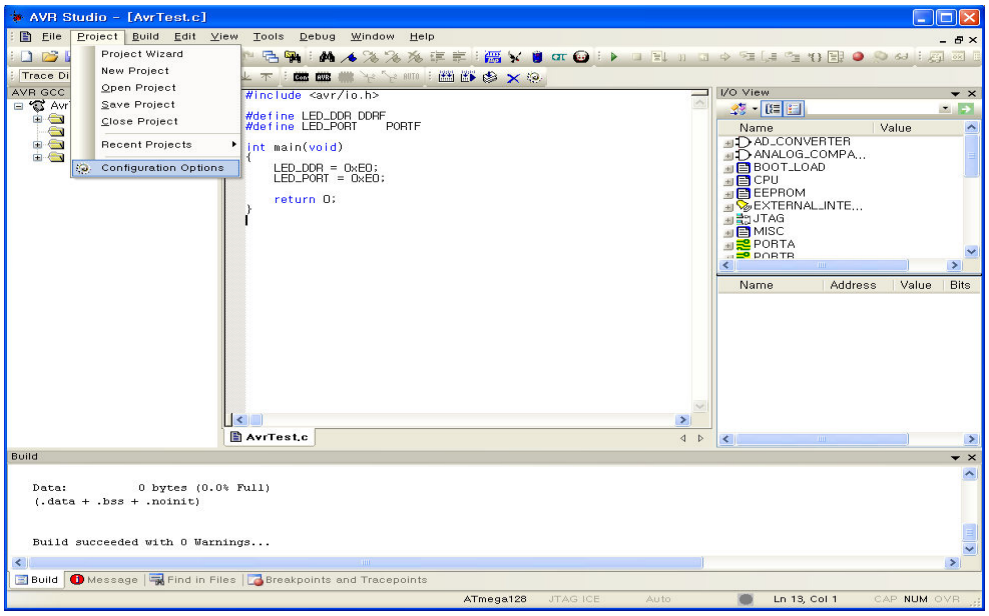
컴파일이 실패했을 때의 화면을 나타내고 있다. 노란색 표시가 Warning, 빨간색 표시가 Error이며 main 함수의 리턴 타입이 int 타입이 아니라는 경고와 count라는 변수가 선언되지 않고 사용되었다는 에러를 표시하고 있다.



1.5 AVR Studio4에서 Program 하기

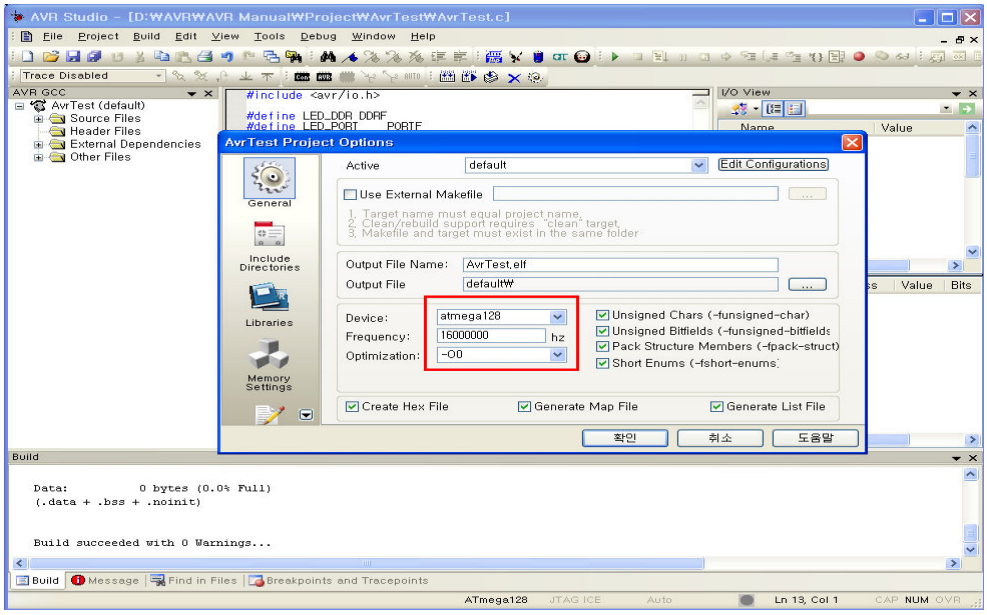
AVR Studio4에서 컴파일 한 결과 파일을 직접 실습 보드에 Download 해 보자. 우선 위의 컴파일 결과로 프로젝트 폴더 내에 'default' 라는 새로운 폴더가 생성되고, 그 아래에 AVRTest1.hex 파일이 생성된 것을 확인한다. 만일 hex 파일이 생성되지 않았다면 컴파일이 제대로 수행되지 않은 것이므로 다시 컴파일 한 후 실습하도록 한다.

다음 그림과 같이 [Project]-> [Configuration Options] 메뉴를 선택한다.

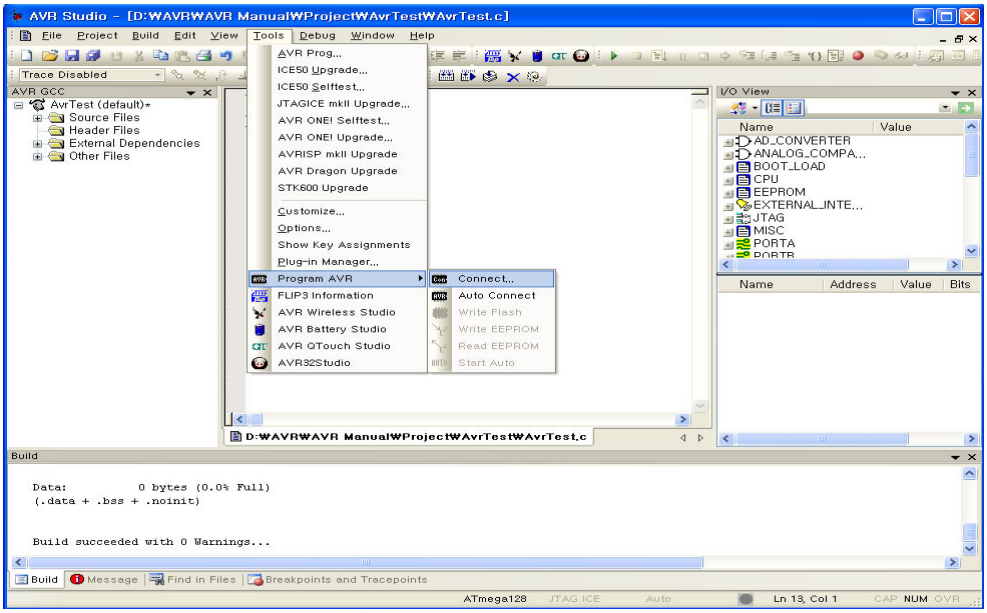


'General' 탭의 항목을 다음과 같이 설정한 후 '확인' 버튼을 클릭하여 저장한다.

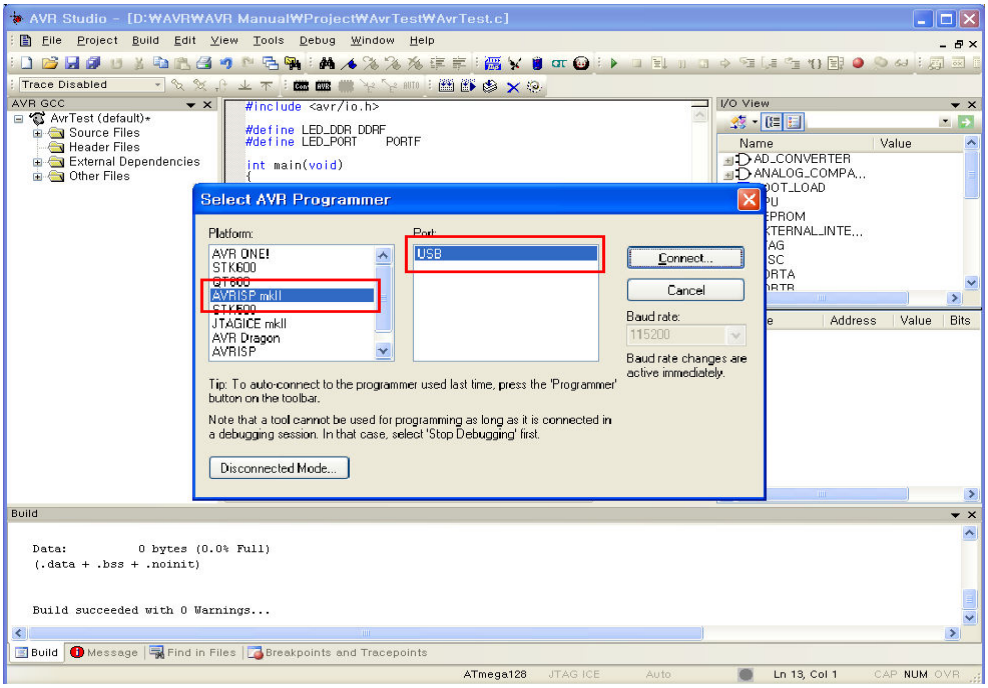
- Freauency : 16000000 hz , CPU Clock을 설정한다.
- Optimization : -O0 , 컴파일 최적화 단계를 설정한다.



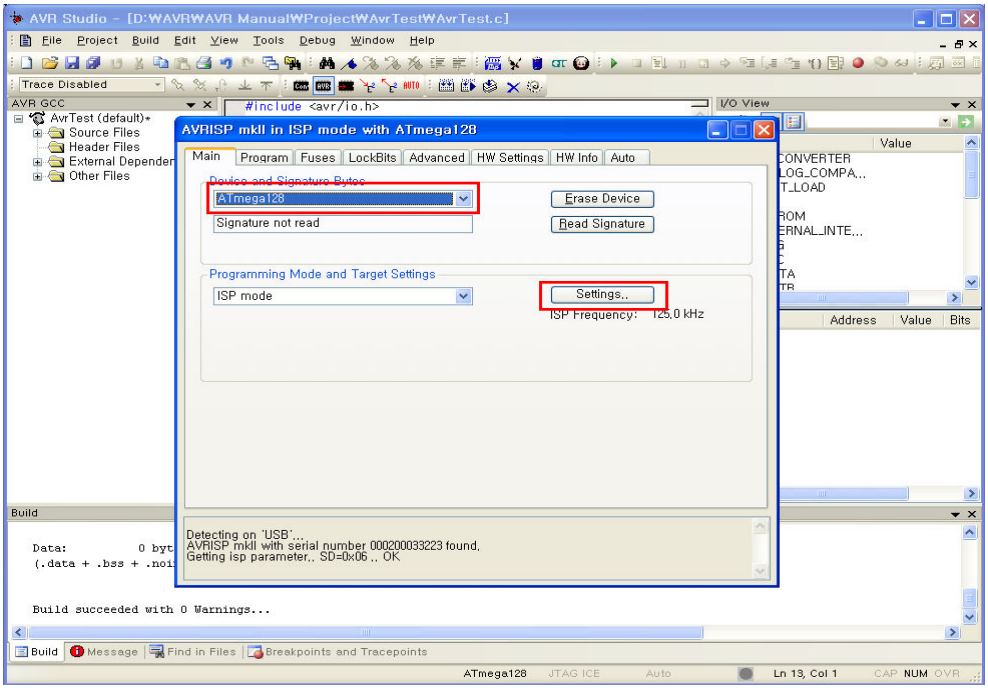
실습 보드의 USB 커넥터와 PC의 USB 포트를 케이블로 연결한 후 실습을 진행한다. [Tools]-> [Program AVR]-> [Connect..] 메뉴를 선택한다. 또는 상단의 'Connect' 아이콘을 선택한다.



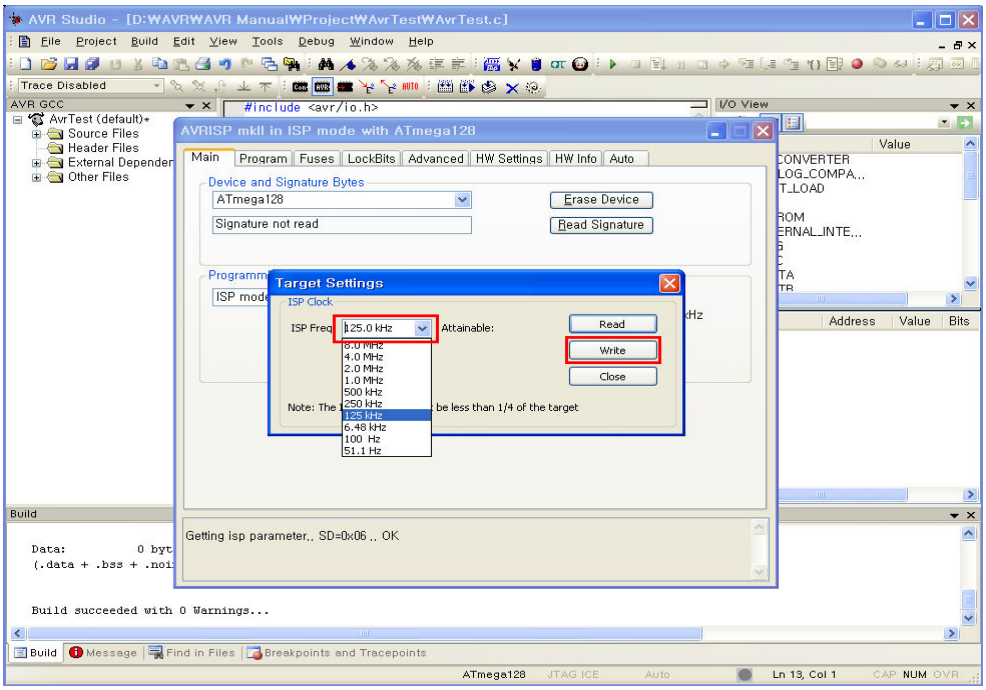
다음 그림과 같이 'AVRISP mkII'를 선택 후 'USB'를 선택한다. 'Connect..' 버튼을 클릭한다.



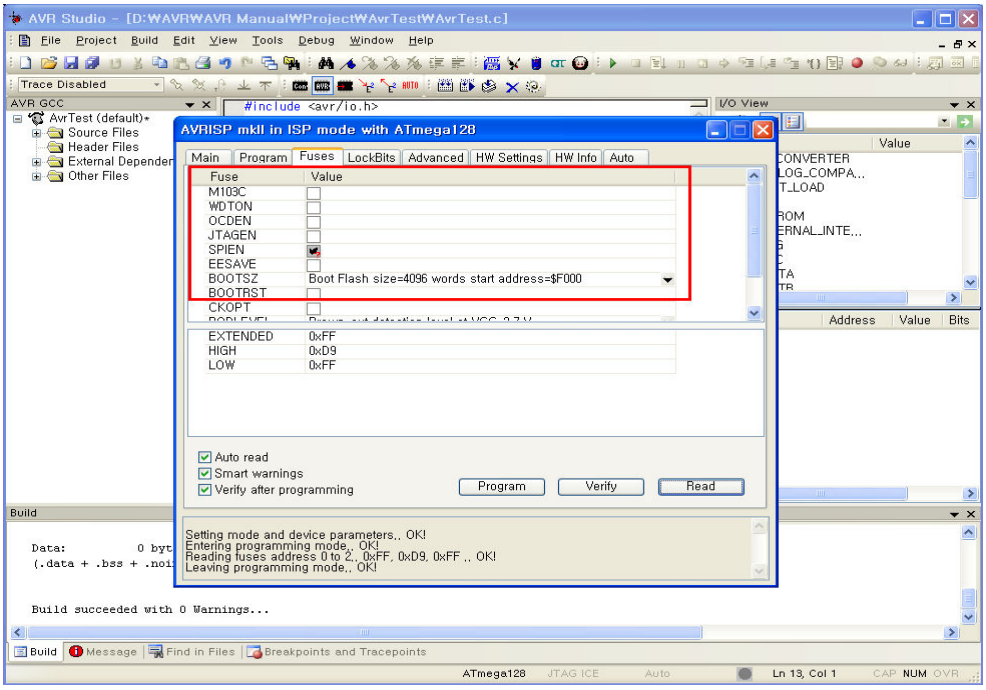
[Main] 탭에서 Device가 'ATmega128' 인 것을 확인 후 'Settings..' 버튼을 클릭한다. 전송 속도를 설정하는 과정이다.



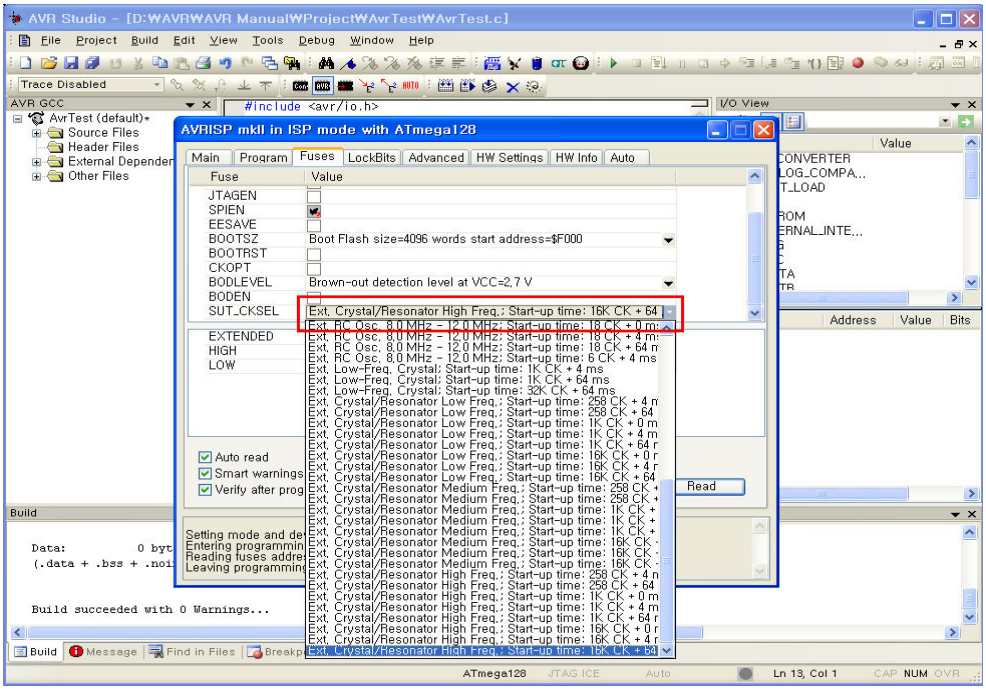
'125khz' 를 선택한 후 'Write' 버튼을 클릭한 후, 'Close' 버튼을 클릭하여 저장한다.



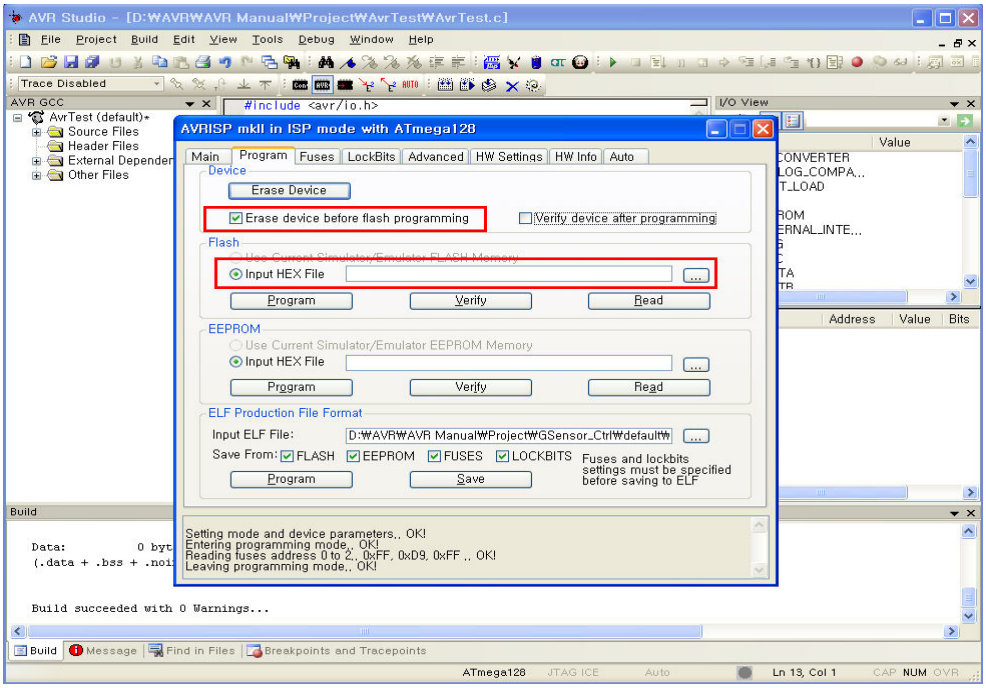
[Fuses] 탭을 선택한 후 다음 그림과 같이 'SPIEN' 외의 항목은 모두 체크 해제한다. Fuses bit의 경우 한번 잘못 설정되면 결과를 돌이키기 힘들기 때문에 주의해야 한다.



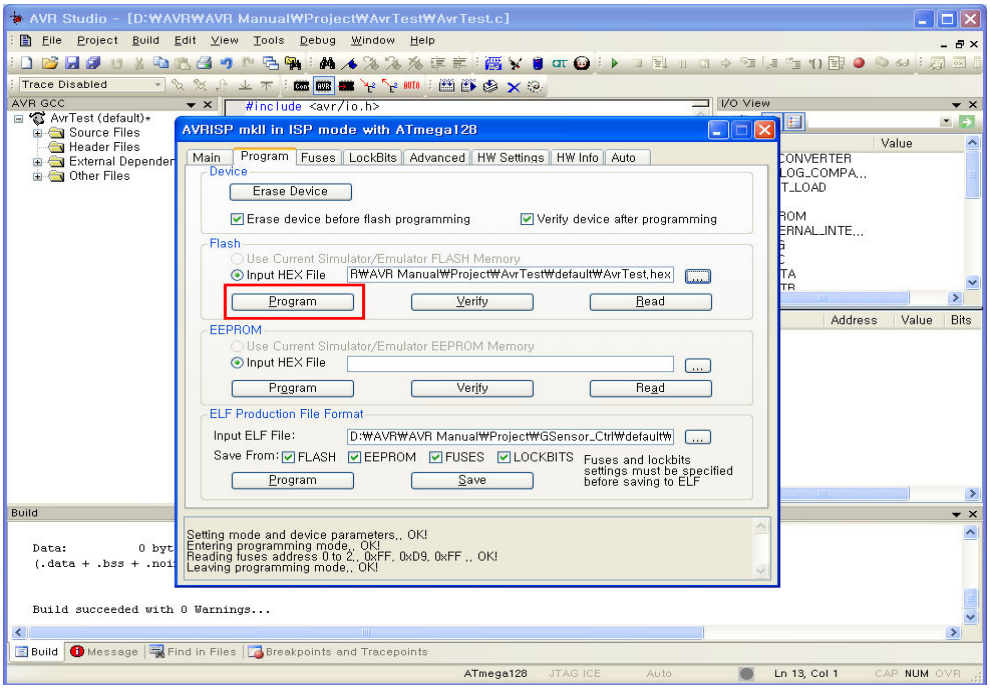
SUT_CKSEL 항목에서 맨 아래의 'Ext. Crystal/Resonator High Freq.; Start-up time: 16K CK + 64'를 선택한다. 선택 후 아래의 'Program' 버튼을 클릭하여 저장한다.



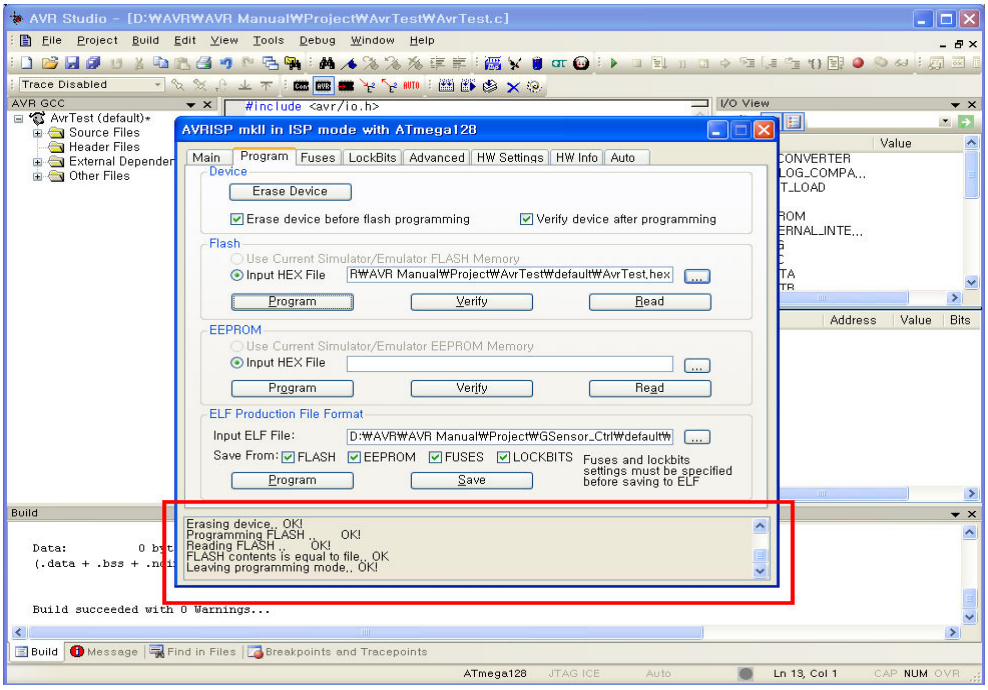
[Program] 탭으로 이동한다. ‘Erase device before flash programming’ 항목에 체크한다. 실행 파일을 메모리에 쓰기 전, 이전 Data를 지운다. ‘Flash’ 항목에서 Program 할 hex 파일을 선택한다.



프로젝트 폴더가 있는 곳으로 이동하여 ‘프로젝트폴더\AVRTest1\default\AVRTest1.hex’ 파일을 선택한다. 선택 후 ‘Program’ 버튼을 클릭하면 다운로드가 시작된다.



다음 그림과 같이 화면이 출력되면 다운로드가 정상적으로 완료된 것이다.

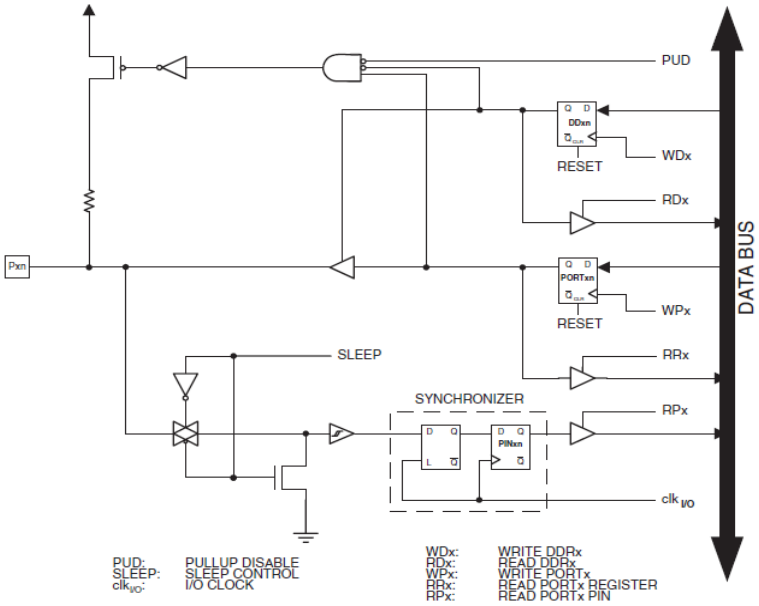


실습 보드의 LED가 모두 켜지는 것을 확인할 수 있을 것이다. 지금까지 컴파일러와 개발 툴을 사용하는 방법에 대해 알아보았다. 이제부터 실제로 실습 보드를 제어하는 실습을 해보도록 하자.

실습 2. GPIO 제어하기

2.1 GPIO(General Purpose Input/Output) 개요

모든 AVR 포트는 범용 디지털 I/O 포트로 사용될 때 Read, Modify, Write 동작이 가능하다. 각 핀은 LED를 직접 켤 수 있을 정도의 전류를 흘려보낼 수 있다. 각 I/O 핀에는 보호용 다이오드가 붙어있으며 선택적으로 Pull-up 저항을 설정하여 사용할 수 있는 양방향 입출력 포트이다. 다음 그림은 General Digital I/O Block Diagram이다.



ATmega128은 총 7개의 GPIO Block을 가지고 있으며 GPIOA~G로 나타낸다. 각 포트는 다음과 같이 3개의 레지스터를 이용하여 bit 단위로 제어할 수 있다.

다음 설명에서 x는 포트 이름(A~G)을 의미한다.

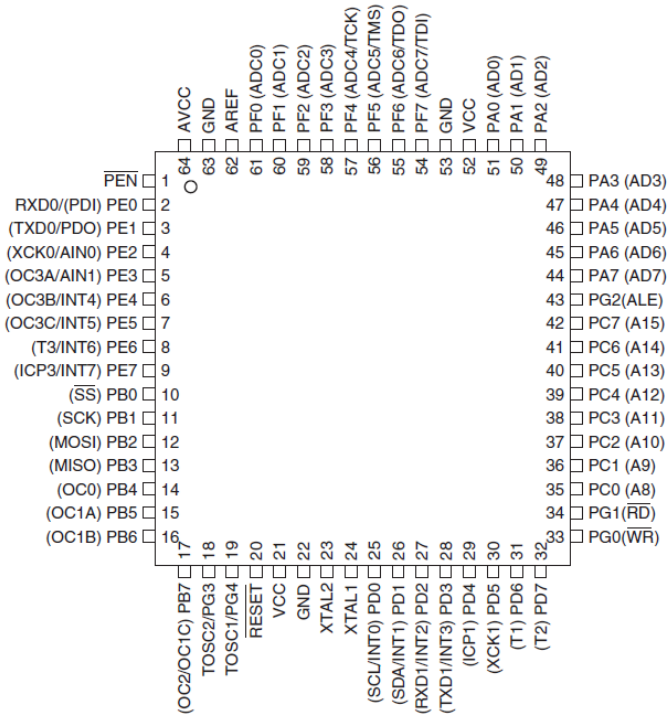
- DDRx (Data Direction Register) : 포트 핀(7~0)의 방향을 결정하며, '1' 이 설정되면 출력 기능을, '0' 이 설정되면 입력 기능을 한다.
- PORTx (Data Register) : 출력 기능으로 설정된 포트 핀으로 데이터를 출력하는데 사용한다.
- PINx (Port Input Pins Register) : 포트 핀의 상태를 읽어올 때 사용한다. 포트가 출력 기능으로 설정되어 있어도 포트 핀의 상태를 읽어올 수 있다.

다음은 각 레지스터의 기능을 간략하게 표로 나타낸 것이다.

레지스터 이름	기능	R/W 여부
DDRx (Data Direction Register)	입출력 방향 설정 각 비트로 구분하여 설정 가능 (0:입력, 1:출력)	R/W 가능
PORTx (Data Register)	데이터 출력용	R/W 가능
PINx (Port Input Pins Register)	데이터 입력용	R만 가능

대부분의 병렬 I/O 포트에 있는 비트들은 범용 I/O 핀으로서의 기능 이외에도 부수적인 기능(alternate function)을 가지고 있다. 아래 그림은 ATmega128 칩의 핀 구조이다. 각 핀의 번호 및 기능을 나타내고 있다. 앞에서 설명했듯이 ATmega 128 칩은 PA부터 PG까지 7개의 GPIO Block을 가지고 있으며, PG를 제외한 각 블록은 8개의 Pin을 가지고 있다. Pxn 옆쪽의 () 안에 있는 내용이 그 핀이 가지는 I/O 기능 외의 부수적인 기능을 나타낸다. 예를 들어 6번 핀의 경우 I/O 기능으로 사용될 경우에는 PortE의 4번째 핀으로 사용되지만, 그 외에 외부 인터럽트를 받아 알려주는 기능으로도 사용될 수 있다.

자세한 내용은 데이터 시트에서 참조할 수 있다.

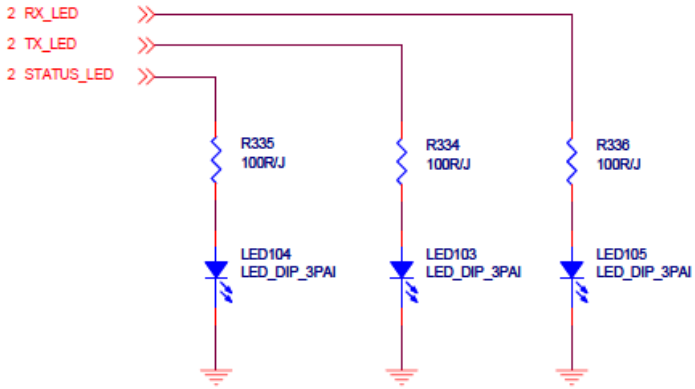


2.2 GPIO 제어 실습하기

GPIO 포트를 이용하여 LED를 제어하는 코드를 작성해 보자. LED를 제어하기 위해서는 실습 보드의 LED가 ATmega128 칩의 어느 핀에 연결되어 있는지 회로도 상에서 확인해야 한다. 지금 우리가 사용하고 있는 실습 보드의 회로도를 살펴보면 LED는 포트E의 2, 5, 6번 핀에 연결되어 있다.

LED가 Vcc나 GND 중 어디로 연결되어 있느냐에 따라 LED를 제어하는 I/O 포트의 값이 달라질 수 있다. LED가 Vcc에 연결되어 있을 경우 I/O 포트에 0(Low)이 설정되어야 LED가 켜지며, 1(High)이 설정되면 꺼진다. 반대로 LED가 GND에 연결되어 있을 경우 I/O 포트에 1(High)이 설정되어야 LED가 켜지며, 0(Low)이 설정되면 LED가 꺼지게 된다.

실습 보드의 경우 아래 그림과 같이 LED가 GND에 연결되어 있으므로, LED를 켜기 위해서는 I/O 포트에 1(High)을 설정해야 한다.



LED를 제어하기 위해서는 DDRx가 출력으로 설정되어야 하며, PORTx에 설정하는 값에 따라 LED가 On/Off 된다. 실습 보드에서 LED가 포트 E의 2, 5, 6번에 연결되어 있으므로 DDRE와 PORTE를 사용하여 LED를 제어할 수 있다.

실습 1. LED On/Off 반복하기

▣ GPIO_LED1.c (Project Name - GPIO_LED1)

```
#include <avr/io.h>

/* ========== LED 제어를 위한 정의 ========== */
#define   DDR_LED        DDRE
#define   PORT_LED       PORTE
/* ==================================== */

/* ============ Main 함수 구현 ============== */
```

```
int main(void)
{
        /* 1. Dummy for문을 위한 변수 선언 */
        int i, j;

        /* PortE 2,5,6번 핀을 출력 기능으로 설정하기 위해 */
        /* DDRE의 2,5,6번 Bit를 1로 Set (0x64 = 01100100b) */
        DDR_LED = 0x64;

        /* PORTE의 모든 Bit를 0으로 초기화  */
        PORT_LED = 0x0;

        /* 반복문을 통해 LED On/Off하기 */
        while(1)
        {
                PORT_LED = 0x64;                /* LED On */

                for(i=0; i<1000; i++)
                        for(j=0; j<1000; j++) ;         /* Delay를 위한 Dummy for 문 */

                PORT_LED = 0x0;                 /* LED Off */

                for(i=0; i<1000; i++)
                        for(j=0; j<1000; j++) ;         /* Delay를 위한 Dummy for 문  */
        }
        return 0;
}
```

※ 참고사항

AVR Studio4에서 컴파일 시 최적화 과정으로 인해 Dummy for문이 무시될 수 있다. 이럴 경우 메뉴의 [Project]-> [Configuration Options]에서 General-> Optimization 설정 값을 '-O0' 로 변경하면 된다.

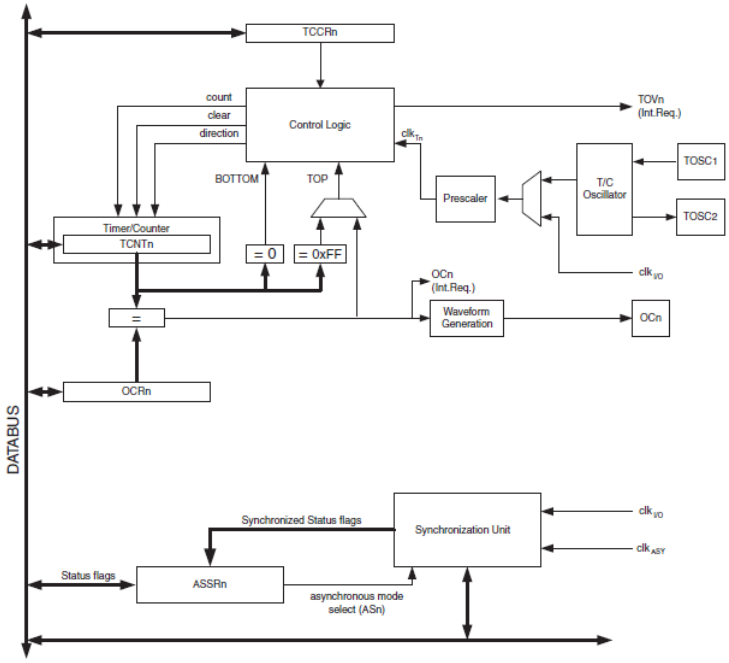
위의 예제에서는 이해를 돕기 위해 0x64라는 16진수 값을 직접 입력하였지만, 다음 실습부터는 가능하면 직접 값을 대입하는 코드는 피하도록 한다. PortE의 2, 5, 6번을 제외한 남은 핀들이 다른 용도로 사용되고 있을 경우 위의 코드가 영향을 끼칠 수 있기 때문이다. 따라서 레지스터의 다른 비트가 다른 용도로 쓰일 가능성이 있을 경우, 해당 비트를 Set 또는 Clear하기 위해서 다음 코드와 같은 방식으로 사용하도록 한다.

```
/* PORT_LED = 0x64; */
PORT_LED |= 0x64;            /* LED On, 2, 5, 6 bit만 Set. 또는, */
PORT_LED |= (1<<5) | (1<<6); /* LED On, 5, 6번 bit만 Set  */

/* PORT_LED = 0x0; */
PORT_LED &= ~(0x64);             /* LED Off, 2, 5, 6 bit만 Clear. 또는, */
PORT_LED &= ~((1<<5) | (1<<6));  /* LED Off, 5, 6번 bit만 Clear */
```

실습 3. Timer/Counter 제어하기

3.1 Timer/Counter 개요

타이머/카운터는 내부 I/O Clock이나 외부의 크리스털을 소스로 사용하고 사용자가 필요로 하는 주기를 설정하여, 일정한 주기로 발생하는 임의의 작업을 할 수 있게 한다. 다음은 ATmega128의 8bit Timer/Counter의 Block Diagram이다.



ATmega128은 Timer/Counter0~3까지를 제공하고 있으며, 여기서는 기본적으로 사용되는 Timer/Count0의 활용 방법에 대해서 알아보기로 한다. Timer/Counter0은 일반적으로 쓰이는 Single 채널, 8 bit Timer/Counter 모듈이다.

특징은 다음과 같다.
- Single 채널 카운터이다.
- 비교하여 일치되면 타이머가 Clear 된다.
- 10 bit의 Clock Prescaler가 있다.
- Overflow와 Compare match의 인터럽트가 있다.
- I/O Clock과 독립된 외부의 32kHz 크리스털로 동작할 수 있다.

Single 채널은 인터럽트가 발생되는 비교 대상이 1:1이라는 것이다. 앞으로 설명할 카운터 레지스터와 비교 레지스터의 값을 비교하여 일치되면 카운터 레지스터의 값은 자동으로 Clear 된다. 적당한 주기를 만들기 위하여 분주비(Prescaler)를 이용하며 총 10 bit인 1024까지 가능하다. 인터럽트가 발생하는 경우는 8 bit 레지스터의 값이 Overflow 되는 경우와, 두 개의 비교 레지스터 값이 일치되는 경우이다. 내부 Clock과 외부 Clock 중에 선택하여 동작시킬 수 있다.

Timer/Counter0은 다음과 같은 4개의 모드를 가진다.
- 타이머 모드 (Normal Mode)
- CTC 모드 (Clear Timer on Compare Match Mode)
- 고속 PWM 모드 (Fast PWM Mode)
- 위상 변경 PWM 모드 (Phase Correct PWM Mode)

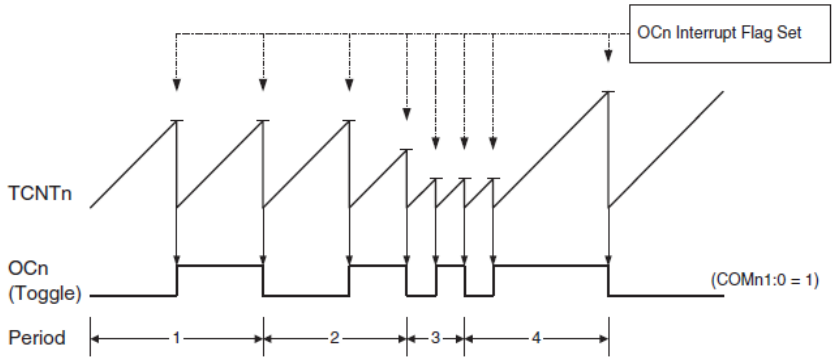
1. 타이머 모드 (Normal Mode)

일반적인 방법으로 클록이 있을 때마다 8bit 레지스터(TCNT0)의 값이 증가하고, 최대값(0xFF)이 되면 다시 '0'으로 Clear 되며, 이때 Overflow Flag(TIFR[TOV0])가 '1'이 되어 Overflow 인터럽트를 요청한다.

2. CTC 모드 (Clear Timer on Compare Match Mode)

클록이 있을 때마다 8bit 레지스터(TCNT0)의 값이 증가하고, 비교 레지스터(OCR0)의 값과 비교해서 일치하면 비교 일치 인터럽트를 요청한다. 이 때 8bit 레지스터(TCNT0)의 값은 '0'으로 Clear 된다.

다음은 Timer/Counter0의 CTC 모드 타이밍도이다.



단, 비교 레지스터(OCR0)의 값을 변경할 때, TCNT0의 현재 값보다 작은 값을 써 넣으면 비교 일치 동작을 놓칠 수 있으므로 주의하여야 한다.

3. 고속 PWM 모드, 위상 변경 PWM 모드

PWM은 Pulse Width Modulation으로 지금은 설명하지 않고, 이 후 Step Motor 제어 실습 때 자세히 설명하겠다.

3.2 Timer/Counter 레지스터

활용 방법에 앞서 주로 사용하는 Timer/Counter 관련 레지스터들에 대해서 알아보자. 실습에서 자주 사용할 레지스터들에 대해 설명할 것이며, 자세한 내용은 데이터 시트를 참조한다.

1. TCCR0 (Timer/Counter0 Control Register)

Timer/Counter0의 Clock 선택 및 모드를 설정한다.

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

- FOC0 (Force Output Compare) : WGM 비트가 PWM 모드가 아닐 경우 유효하며, Write만 가능하다.
  - '1' 을 써 넣는 즉시 파형 발생 Unit이 강제로 동작하여, COM01,00에 설정된 대로 OC0로 출력한다.
  - 이 비트로 인한 인터럽트는 발생되지 않으며, CTC 모드에서 OCR0가 TOP이 되어도 타이머는 Clear 되지 않는다.
- WGM00, WGM01 (Waveform Generation Mode) : 파형 출력 기능을 설정하기 위한 비트이다.

Mode	WGM01(1) (CTC0)	WGM00(1) (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

- Normal Mode는 Overflow 인터럽트를 사용하는 기본적인 모드이다.
- 비교 인터럽트를 사용하기 위해서는 CTC Mode로 설정한다. CTC Mode로 설정했을 경우 위의 표와 같이 TOP이 0xFF가 아닌 OCR0 값이 된다.
- PWM Mode는 Step Motor 제어에서 자세히 다루도록 하겠다.

- COM01, COM00 (Compare Match Output Mode) : OC0의 출력을 제어한다. WGM01,00에 따라 결정된다.
  - PWM Mode가 아닐 경우

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

0 : 포트로 동작하며, OC0은 사용하지 않는다.
1 : TCNT0과 OCR0의 값이 일치하면 OC0 값을 toggle 시킨다.
2 : TCNT0과 OCR0의 값이 일치하면 OC0 값을 '0' 으로 Clear 한다.
3 : TCNT0과 OCR0의 값이 일치하면 OC0 값을 '1' 로 Set 한다.

- CS02, CS01, CS00 (Clock Select) : 사용할 클록과 분주비(Prescaler)를 선택한다.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T0S}/(No prescaling)
0	1	0	clk_{T0S}/8 (From prescaler)
0	1	1	clk_{T0S}/32 (From prescaler)
1	0	0	clk_{T0S}/64 (From prescaler)
1	0	1	clk_{T0S}/128 (From prescaler)
1	1	0	clk_{T0S}/256 (From prescaler)
1	1	1	clk_{T0S}/1024 (From prescaler)

2. TCNT0 (Timer/Counter Register)

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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	

하드웨어에 의해 자동으로 증가하는 8 bit 카운터 값이다. 카운터가 동작하는 동안 TCNT0 값을 변경하는 것은 TCNT0과 OCNT0의 비교에 실패할 위험이 있다.

3. OCR0 (Output Compare Register)

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
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	

카운터 값인 TCNT0과 비교가 되는 8 bit 값이다. 클록마다 비교하여 비교 인터럽트나 OC0 핀으로 파형을 내보내기 위해 사용된다.

4. TIMSK (Timer/Counter Interrupt Mask Register)

Timer/Counter의 인터럽트 요청을 제어하며, Timer/Counter0에서 사용하는 기능만 살펴본다.

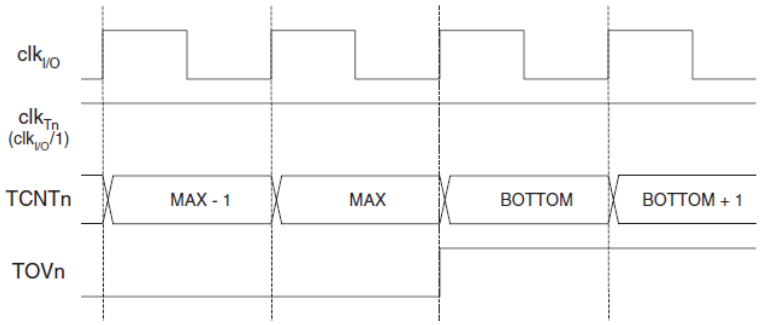
Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- OCIE0 (Timer/Counter0 Output Compare Match Interrupt Enable) : 비교 일치 인터럽트를 허용한다. 이 비트가 '1' 로 되어 있고, 상태 레지스터(Status Register)의 I 비트가 '1' 로 되어 있으면 비교 일치 인터럽트가 활성화된다.
- TOIE0 (Timer/Counter0 Overflow Interrupt Enable) : Overflow 인터럽트를 허용한다. 이 비트가 '1' 이고, 상태 레지스터(Status Register)의 I 비트가 '1' 로 설정되면 Overflow 인터럽트가 활성화된다.

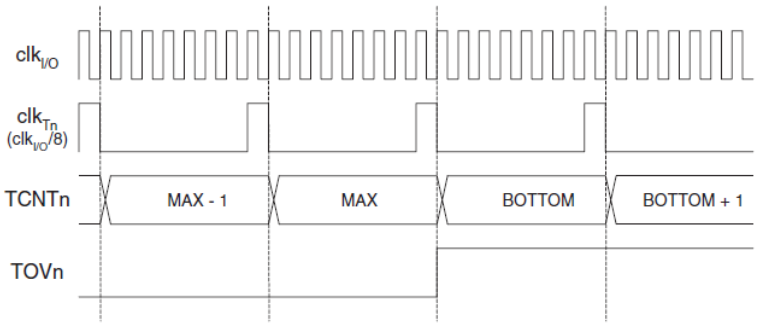
3.3 Timer/Counter 제어 실습하기

앞에서 설명한 레지스터들을 이용하여 주기를 설정하고 해당 인터럽트를 처리할 수 있다. 다음 그림은 내부 I/O Clock을 사용하고 분주비(Prescaler)를 사용하지 않을 때와, 8로 설정했을 때 TCNT0의 값이 최대값(Max)에서 최소값(Bottom)으로 바뀌는 상태에 대한 타이밍도이다.

- 분주비(Prescaler)를 사용하지 않을 경우



- 분주비(Prescaler)가 '8' 일 경우



clk I/O는 내부 I/O Clock을 말한다. 아래의 clk Tn은 clk I/O를 분주비(Prescaler)로 나눈 만큼의 주기를 갖는 Clock이 된다. 그리고 TCNT0이 최대값(MAX, 일반적으로 255)이 되어 Overflow가 발생하여 최소값(BOTTOM, 일반적으로 0)으로 바뀐 후 다시 증가하는 모습을 보여 준다. 그 아래에 나오는 TOV0은 Timer/Counter0 Overflow Flag로, Overflow 인터럽트가 발생했을 때 '1' 로 설정되며 인터럽트 핸들링 벡터가 실행되면 하드웨어에 의해 Clear 된다. 즉, 위 그림에서 TOVn이 1로 설정되는 부분이 Overflow 인터럽트가 발생하는 시점이다.

그렇다면, 원하는 시간 간격마다 인터럽트가 발생하게 하는 방법에 대해 생각해보자. 예를 들어 1ms마다 Overflow 인터럽트가 발생하도록 설정해보자. ATmega128의 내부 I/O Clock은 16MHz이다. 즉, 1초에 16000000번 clk I/O가 발생하므로 1ms에는 '16000000/1000' 만큼 clk I/O가 발생할 것이다. 분주비를 64로 설정한다면 1ms에 '16000000/1000/64' 만큼 clk Tn이 발생하고, 그 값만큼 TCNT0의 값이 증가한다. (분주비 64는 TCNT0이 255보다 작게 나오게 하기 위해 임의로 설정한 값이다.)

TCNT0이 '16000000/1000/64' 만큼 증가했을 때 Overflow 인터럽트가 발생하게 하기 위해 TCNT0의 초기 값을 다음과 같은 공식으로 설정할 수 있다.

```
TCNT0 = 256 - (CPU_CLOCK / TICK_PER_SEC / PRESCALER)
```

실습 1. Overflow 인터럽트를 이용한 Timer 제어하기

Overflow 인터럽트를 이용하여 Delay_ms() 함수를 만들고 LED를 1초 간격으로 깜빡이도록 하라.

▣ TIMER_Overflow.c (Project Name - TIMER_Overflow)

```
#include <avr/io.h>
#include <avr/interrupt.h>              /* Interrupt 관련 Header file */

/* ================= LED 제어를 위한 정의 ================== */
#define   DDR_LED DDRE
#define   PORT_LED          PORTE
/* ==================================================== */

/* ================= Interrupt 제어를 위한 정의 ============== */
/* 1ms 단위로 인터럽트를 발생시키기 위해 TCNT0 값을 정의 */
#define   CPU_CLOCK         16000000        /* CPU Clock = 16MHz */
#define   TICK_PER_SEC      1000            /* 1 Sec = 1000 ms */
#define   PRESCALER         64              /* Prescaler = 64 */
#define   TCNT_VALUE        (CPU_CLOCK/TICK_PER_SEC/PRESCALER)
/* ==================================================== */

/* =========== Interrupt 제어를 위한 전역변수 선언============== */
volatile unsigned int tickCnt;
/* ==================================================== */


/* ==================== Delay 함수 선언 ==================== */
void Delay_ms(unsigned int msec);
/* ==================================================== */

/* ==================== Main 함수 구현 ==================== */
int main(void)
{
        /* LED 출력을 위한 DDRE, PORTE 초기화 */
        DDR_LED |= 0x64;
        PORT_LED &= ~(0x64);

        /* Timer/Counter0 제어 레지스터 설정 */
        /* Normal Mode, Not used OC0, 64 Prescale */
        TCCR0 = (1<<CS02);

        /* TCNT0 초기 값 설정  */
```

```
        /* TCNT0 = 256 - (16000000 / 1000 / 64); */
        TCNT0 = 256 - TCNT_VALUE;

        /* Overflow Interrupt Enable  */
        TIMSK = (1<<TOIE0);

        /* Set Enable Interrupt (Status Register) */
        sei();

        while(1)
        {
                PORT_LED |= 0x64;           /* 4. LED On 후 1초간 Delay */
                Delay_ms(1000);

                PORT_LED &= ~(0x64);        /* 5. LED Off 후 1초간 Delay */
                Delay_ms(1000);
        }

        return 0;
}
/* ================================================== */

/* ================== Delay 함수 구현 ================== */
void Delay_ms(unsigned int msec)
{
        tickCnt = 0;                        /* 전역변수 초기화 */
        while(tickCnt < msec) ;             /* tickCnt가 msec만큼 증가할 때까지 기다림 */
}
/* ================================================== */

/* ============== Interrupt Handler 함수 구현 =============== */
/* SIG_OVERFLOW0 인터럽트 발생 시 Interrupt Handling Vector에 의해 호출 */
SIGNAL(SIG_OVERFLOW0)
{
        tickCnt++;                          /* Interrupt 발생 시 tickCnt 증가 */
        TCNT0 = 256 - TCNT_VALUE;           /* TCNT0 값을 다시 초기화 */
}
/* ================================================== */
```

※ 참고사항

위의 소스에서 sei() 함수는 실제로 인터럽트를 사용하기 위해서 글로벌 인터럽트를 활성화시키는 라이브러리 함수이며, 반대로 cli() 함수는 글로벌 인터럽트를 비활성화 시키는 함수이다. 두 함수는 <avr/interrupt.h> 파일에 선언되어 있다.

SIGNAL() 함수 역시 sei() 함수와 함께 <avr/interrupt.h> 파일에 선언되어 있는 라이브러리 함수이다. 인터럽트가 발생했을 경우 하드웨어에 의해 호출되는 인터럽트 벡터 함수이며, 리턴 타입이 없다. 모두 앞으로의 실습에 자주 사용되는 라이브러리 함수이므로 꼭 기억해 두도록 한다.

실습 2. Compare Match 인터럽트를 이용하여 Timer/Counter 제어하기

Compare Match 인터럽트를 이용하여 Delay_ms() 함수를 만들고 LED를 1초 간격으로 깜빡이도록 하라.

▣ TIMER_CTC.c (Project Name - TIMER_CTC)

```
#include <avr/io.h>
#include <avr/interrupt.h>

/* ================= LED 제어를 위한 정의 ================== */
#define   DDR_LED          DDRE
#define   PORT_LED         PORTE
/* ==================================================== */

/* =============== Interrupt 제어를 위한 정의 ================ */
/* 1ms 단위로 인터럽트를 발생시키기 위해 TCNT 값을 정의  */
#define   CPU_CLOCK        16000000        /* CPU Clock = 16MHz */
#define   TICK_PER_SEC     1000            /* 1 Sec = 1000 ms  */
#define   PRESCALER        64              /* Prescaler = 64 */
#define   TCNT_VALUE       (CPU_CLOCK/TICK_PER_SEC/PRESCALER)
/* ==================================================== */

/* ============  Interrupt 제어를 위한 전역변수 선언 ============ */
volatile unsigned int tickCnt;
/* ==================================================== */

/* ================= Delay 함수 선언 ===================== */
void Delay_ms(unsigned int msec);
/* ==================================================== */

/* ================== Main 함수 구현 ==================== */
int main(void)
{
        /* LED 출력을 위한 DDRE, PORTE 초기화 */
        DDR_LED |= 0x64;
```

```
        PORT_LED &= ~(0x64);

        /* Timer/Counter0 제어 레지스터 설정 */
        /* CTC Mode, Not used OC0, 64 Prescale */
        TCCR0 = (1<<WGM01) | (1<<CS02);

        TCNT0 = 0;              /* TCNT0 초기 값 설정    */
        OCR0 = TCNT_VALUE;      /* OCR0 값 설정   */

        /* Compare Match Interrupt Enable  */
        TIMSK = (1<<OCIE0);

        /* Set Enable Interrupt (Status Register)  */
        sei();

        while(1)
        {
                PORT_LED |= 0x64;           /* 4. LED On 후 1초간 Delay  */
                Delay_ms(1000);

                PORT_LED &= ~(0x64);        /* 5. LED Off 후 1초간 Delay  */
                Delay_ms(1000);
        }
        return 0;
}
/* ================================================== */

/* ================== Delay 함수 구현 ================== */
void Delay_ms(unsigned int msec)
{
        tickCnt = 0;                        /* 전역변수 초기화 */
        while(tickCnt < msec) ;             /* tickCnt가 msec만큼 증가할 때까지 기다림 */
}
/* ================================================== */

/* ============== Interrupt Handler 함수 구현 ============== */
/* SIG_OUTPUT_COMPARE0 인터럽트 발생시 Interrupt Handling Vector에 의해 호출 */
SIGNAL(SIG_OUTPUT_COMPARE0)
{
        tickCnt++;                          /* Interrupt 발생 시 tickCnt 증가 */
```

```
        TCNT0 = 0;                          /* TCNT0 값을 다시 초기화 */
}
```

*** 참고사항**

위의 예제처럼 Overflow 인터럽트, 또는 Compare Match 인터럽트를 사용하여 Delay 함수를 만들 수 있지만, 매번 인터럽트 처리 함수를 추가하는 것은 매우 번거로운 일이다. 또한 다른 인터럽트 처리 루틴 내에서 사용할 수 없다는 단점이 있다.

WinAVR에서는 delay.h 라이브러리를 제공하여 '_delay_ms(double _ms)' 함수와 '_delay_us(double _us)' 함수를 제공하고 있으므로, 앞으로의 실습 예제에서는 이 함수를 사용하도록 한다.

단, C:\WinAVR\avr\include\util\delay.h 파일을 열어 CPU Clock이 정의되어 있는 부분을 다음과 같이 수정해야 한다.

```
#ifndef F_CPU
/* prevent compiler error by supplying a default */
# warning "F_CPU not defined for <util/delay.h>"
# define F_CPU 16000000UL
#endif
```

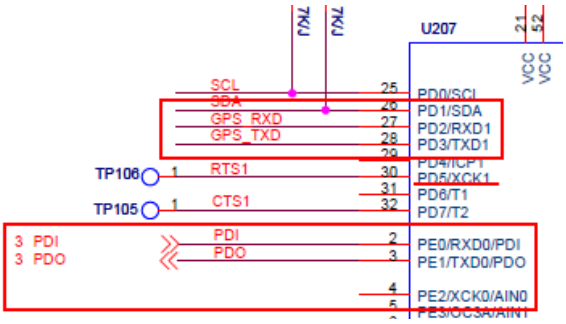
실습 4. USART 통신 제어하기

4.1 USART 통신 개요

ATmega128에서는 USART(Universal Synchronous and Asynchronous serial Receiver and Transmitter)를 지원하고 있는데, 주요 특징은 다음과 같다.

- Full Duplex 동작 (직렬 송수신이 독립적으로 동작)
- 비동기와 동기 모두 동작
- 전송 속도 제어
- 5~9비트 데이터와 1~2비트의 Stop Bit
- 홀수, 짝수 패리티 발생기와 패리티 체크를 하드웨어적으로 지원
- 잘못된 Start Bit 검출과 Digital Low Pass Filter를 포함한 Noise Filtering
- Overrun 및 Framing Error 검출
- 멀티프로세서 통신 모드 지원

ATmega128은 USART0과 USART 1 두 개의 USART를 가지고 있다. 실습 보드에서 USART1은 GPS 센서를 사용하기 위해 연결되어 있으므로, 우리는 USART0을 사용하도록 한다. 다음 그림은 실습 보드의 회로도 중 USART PIN 부분을 나타낸 것이다.

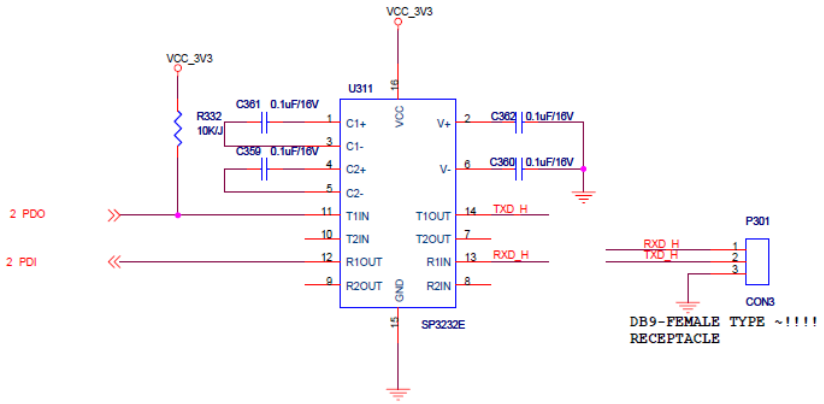


각 PIN의 기능은 다음과 같다.

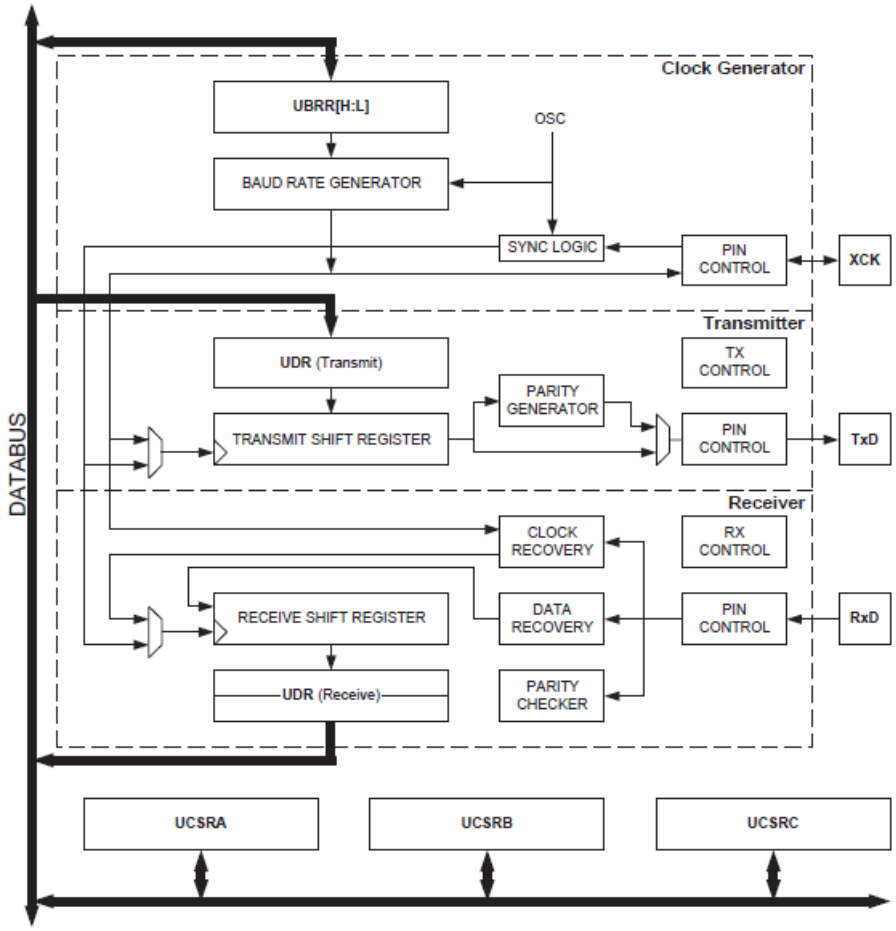
포트 핀	다른 기능
PE0	RXD0(UART0 Receive Pin) : UART0 수신 입력, DDRE0의 값에 관계없음
PE1	TXD0(UART0 Transmit Pin) : UART0 송신 출력
PE2	XCK0(USART0 extern clock input/output) : USART0 동기모드의 클록 입출력
PD2	RXD1(UART1 Receive Pin) : UART1 수신 입력, DDRD2의 값에 관계없음
PD3	TXD1(UART1 Transmit Pin) : UART1 송신 출력, DDRD3의 값에 관계없음
PD5	XCK1(USART1 extern clock input/output) : USART0 동기모드의 클록 입출력

XCKn 핀은 동기 모드로 사용할 경우에 클록 입출력을 위해 사용하며, DDR의 값이 '1' 이면 클록 출력으로, '0' 이면 클록 입력으로 사용한다. 그러나 현재 우리가 실습할 보드에서는 비동기 모드만을 사용할 것이므로 XCKn 핀 기능은 사용하지 않는다.

실습 보드의 USART 커넥터는 병렬 포트 옆에 있는 3PIN 직렬 포트 커넥터이다. 아래 그림은 ATmega128 칩의 2,3번 Pin을 SP3232E를 사용하여 3PIN 커넥터로 연결된 모습을 나타내주는 회로도이다.



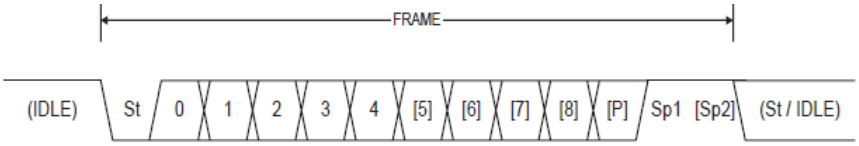
USART는 Clock Generator Unit, Transmitter Unit, Receiver Unit으로 구성되어 있으며, Clock Generator Unit은 비동기 모드 혹은 동기 마스트 모드에 사용할 클록을 계산하여 출력한다. 다음 그림은 USART 내부 구조를 나타내는 Block Diagram이다.



각 레지스터에 대해서는 다음 절에서 설명한다.

4.2 USART 통신 프레임 형식 및 동작 모드

USART 통신에 사용되는 프레임은 다음 그림과 같이 Start Bit(1bit) + Data(5~8bit) + Parity Bit(0~1bit) + Stop Bit(1~2bit) 형식으로 전송된다.



St(Start Bit)는 항상 Low(0)이다. 데이터는 5~7비트일 경우, 송신할 때 상위 비트는 무시되며 수신할 때는 상위 비트가 '0' 으로 채워진다. P(Parity Bit)는 없거나 홀수 또는 짝수를 나타내고, Sp(Stop Bit)는 항상 High(1)로 1비트 또는 2비트로 구성된다.

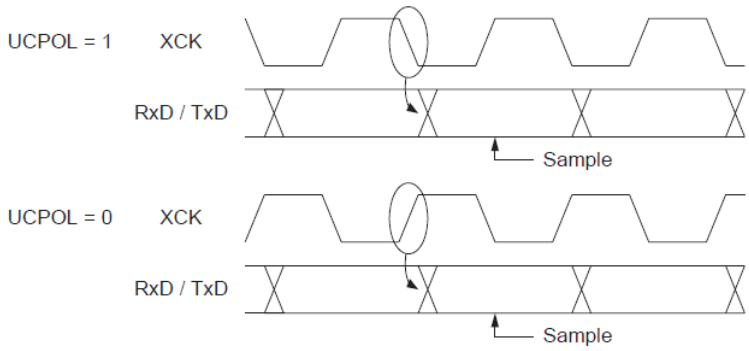
USART는 다음과 같이 3가지 모드를 지원한다.

1. 비동기 모드 (Asynchronous Mode)

일반적인 데이터 통신 방법으로, 앞에서 설명한 프레임 형식으로 통신한다. UCSR 레지스터의 설정에 따라 데이터 크기 및 Parity Bit, Stop Bit의 크기가 달라질 수 있다.

2. 동기 모드 (Synchronous Mode)

마스터 모드에서는 XCK 핀을 출력 기능으로, 슬레이브 모드에서는 입력 기능으로 사용하여 통신한다. 슬레이브 모드에서 XCK 핀으로 입력되는 Clock Frequency는 'System Clock /4' 보다 작아야 한다. 다음은 UCSRC 레지스터의 UCPOL 값에 따라 XCK 클록의 상승/하강 모서리에서 송신 신호의 변화 혹은 수신 신호의 샘플 방법이 달라지는 것을 나타낸 그림이다.



3. 멀티프로세서 모드 (Multi-processor Mode)

멀티프로세서 모드로 설정되면 송신기는 영향을 주지 않지만, 수신기에서는 Address 정보가 포함되지 않은 수신 데이터는 무시되어 수신기 버퍼에 저장되지 않는다.

- 수신기의 데이터 크기가 5~8비트로 설정되어 있을 때, 첫 번째 Stop Bit가 '1' 이면 Address, '0' 이면 데이터를 나타낸다.
- 수신기의 데이터 크기가 9비트로 설정되어 있다면, 9번째 비트가 '1' 이면 Address, '0' 이면 데이터를 나타낸다.

4.3 USART 레지스터

USART 통신 설정에 사용되는 레지스터에 대해 알아보자. 레지스터 이름 뒤에 붙은 'n'은 USART0을 사용하는지, USART1을 사용하는지에 따라 0, 또는 1이 된다.

1. UDRn (USARTn I/O Data Register)

Bit	7	6	5	4	3	2	1	0	
	RXBn[7:0]								UDRn (Read)
	TXBn[7:0]								UDRn (Write)
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	

송수신 레지스터로 실제로 전달되는 송수신 데이터 값이 저장되어 있는 레지스터이다. UDR에 데이터를 저장하고 전송하면 UDR 레지스터의 값이 전송되고, 읽을 준비를 한 뒤 UDR 레지스터의 값을 읽으면 데이터를 수신할 수 있다.

2. UCSRnA (USART Control and Status Register A)

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

UCSR 레지스터들은 A, B, C 세 개로 나뉘어져 있으며, 설정이나 상태와 관련된 레지스터이다. UCSRA의 주요 비트는 다음과 같다. 거론되지 않은 비트들에 대해서는 여기서 다루지 않으며, 자세한 내용은 데이터 시트를 참조한다.

- **RXCn** (USART Receive Complete) : 수신 버퍼에 읽지 않은 데이터가 있는 경우 '1'로 설정되며, 수신 버퍼가 비워지면 '0'이 된다.
- **TXCn** (USART Transmit Complete) : 송신할 데이터가 없는 경우 '1'이 되며, 송신 완료 인터럽트 처리로 들어가거나 '1'을 써넣으면 '0'이 된다.
- **UDREn** (USART Data Register Empty) : 송신 버퍼가 새로운 데이터를 받을 준비가 되었을 경우 '1'이 된다. 송신 가능 여부를 나타낸다.
- **MPCMn** (Multi-processor Communication Mode) : '1'로 설정할 경우 멀티프로세서 모드로 설정되며, Address 정보가 포함되지 않은 수신 데이터는 모두 무시된다. 송신기에서는 사용하지 않는다.

3. UCSRnB (USART Control and Status Register B)

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **RXCIEn** (RX Complete Interrupt Enable) : UCSRA의 RXC 비트를 참조하여 발생하는 수신 인터럽트를 활성화 시킨다.

- **TXCIE** (TX Complete Interrupt Enable) : TXC 비트를 참조하여 발생하는 송신 인터럽트를 활성화 시킨다.
- **RXENn** (Receiver Enable) : USART 수신을 활성화 시킨다. 수신 버퍼와 관련된 동작이 가능하도록 한다.
- **TXENn** (Transmitter Enable) : USART 송신을 활성화 시킨다. 송신 버퍼와 관련된 동작이 가능하도록 한다.
- **UCSZn2** (Character Size) : UCSRC 레지스터의 UCSZn1,0 비트와 함께 송수신에 사용할 데이터의 크기를 설정한다. 자세한 내용은 UCSRC 레지스터에서 설명한다.

4. UCSRnC (USART Control and Status Register C)

Bit	7	6	5	4	3	2	1	0	
	–	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
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	1	1	0	

- **UMSELn** (USART Mode Select) : USART 통신 모드를 설정한다. 설정할 수 있는 값은 다음과 같으며, 여기서는 간단한 저속 직렬 통신을 사용할 것이므로 비동기 방식으로 설정할 것이다.

UMSELn	Mode
0	Asynchronous Operation
1	Synchronous Operation

- **UPMn1,0** (USART Parity Mode) : 패리티 모드를 설정한다. 패리티 모드의 종류와 설정 값은 다음과 같다.

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	(Reserved)
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **USBSn** (USART Stop Bit Select) : Stop Bit를 설정한다. 설정 값은 다음과 같다.

USBSn	Stop Bit(s)
0	1-bit
1	2-bits

- **UCSZn1,0** (USART Character Size) : UCSRB의 UCSZ2 비트와 함께 데이터 비트 크기를 설정한다.

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

5. UBRRnL, UBRRnH (USART Baud Rate Registers)

두 레지스터로 전송 속도를 의미하는 Baud Rate를 설정한다. 레지스터에 설정할 수 있는 값의 위치는 다음과 같다.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

Baud Rate를 계산하는 방법은 다음과 같다.

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

BAUD는 bps(bit per second) 단위의 보율을 의미하며 fosc는 시스템의 오실레이터 주파수이다. 실습 보드에서는 16MHz 크리스털을 사용하고 있다. ATmega128의 데이터 시트에 각 fosc에 대한 UBRR 값을 계산한 도표가 있으므로 참고한다. 다음 그림은 16MHz에서의 UBRR 값이다.

Baud Rate (bps)	f_{osc} = 16.0000 MHz				Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2X = 0		U2X = 1			U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error		UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	38.4k	25	0.2%	51	0.2%
4800	207	0.2%	416	-0.1%	57.6k	16	2.1%	34	-0.8%
9600	103	0.2%	207	0.2%	76.8k	12	0.2%	25	0.2%
14.4k	68	0.6%	138	-0.1%	115.2k	8	-3.5%	16	2.1%
19.2k	51	0.2%	103	0.2%	230.4k	3	8.5%	8	-3.5%
28.8k	34	-0.8%	68	0.6%	250k	3	0.0%	7	0.0%

4.4 USART 통신 제어하기

USART 통신을 위해 필요한 과정에 대해서 먼저 살펴보도록 하자.

1. USART Initialize

초기화 부분에 들어가야 할 내용은 Baud rate 설정과 송수신 활성화, Frame 형식 설정이다. 아래의 예제는 19200bps, No Parity, 1bit Stop Bit, 8bit Data로 설정하는 방법을 나타내고 있다. 또한 USART 수신시에 인터럽트를 사용하기 위해 RXCIE 비트를 1로 설정하여 수신 완료 인터럽트를 활성화 시킨다.

▣ USART 초기화 예제

```
/* =============== USART 제어를 위한 정의 ================ */
#define BAUD_RATE 51
/* ================================================== */

/* =============== USART 초기화 함수 구현 ================ */
void USART_Init(void)
{
        /* Baud Rate 설정 */
        UBRR0H = (unsigned char) BAUD_RATE >> 8;
        UBRR0L = (unsigned char) BAUD_RATE;

        /* Transmitter, Receiver, Receive Complete Interrupt Enable  */
        UCSR0B = (1<<TXEN) | (1<<RXEN) | (1<<RXCIE);

        /* Frame Format 설정  */
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);
}
/* ================================================== */
```

2. USART Transmitter

송신 방법은 간단하게 UCSR0A 레지스터의 UDRE 비트를 체크하여 송신 버퍼가 비워졌는지를 확인하고, 송신이 가능하게 되면 데이터를 UDR0에 채우면 된다. 다음은 간단한 예제이다.

▣ USART 송신 예제

```
/* =============== USART 전송 함수 구현 ================ */
void USART_Transmitter(char data)
{
        /* 버퍼가 비워질 때까지 기다린다. */
        while(!(UCSR0A & (1<<UDRE)) ;
        UDR0 = data;
}
/* ================================================== */
```

3. USART Receiver

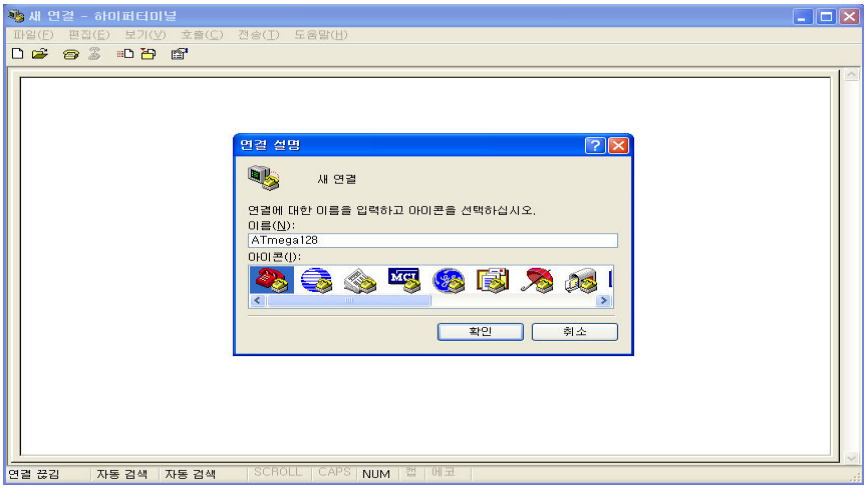
USART를 초기화하는 부분에서 수신 완료 인터럽트를 활성화시켰기 때문에 인터럽트 처리 루틴에 수신 루틴을 넣을 수 있다. UCSR0A 레지스터의 RXC 비트를 살펴보고 수신 버퍼에 아직 읽지 않은 데이터가 있다면 버퍼의 값을 읽어온다. 다음은 간단한 예제이다.

▣ USART 수신 예제

```
/* ============= USART Interrupt 처리 함수 구현 ============= */
SIGNAL(SIG_UART0_RECV)
{
        /* 읽지 않은 데이터가 존재한다면 버퍼의 값을 읽어온다. */
        if(UCSR0A & (1<<RXC))
        {
                data = UDR0;
        }
}
/* ================================================== */
```

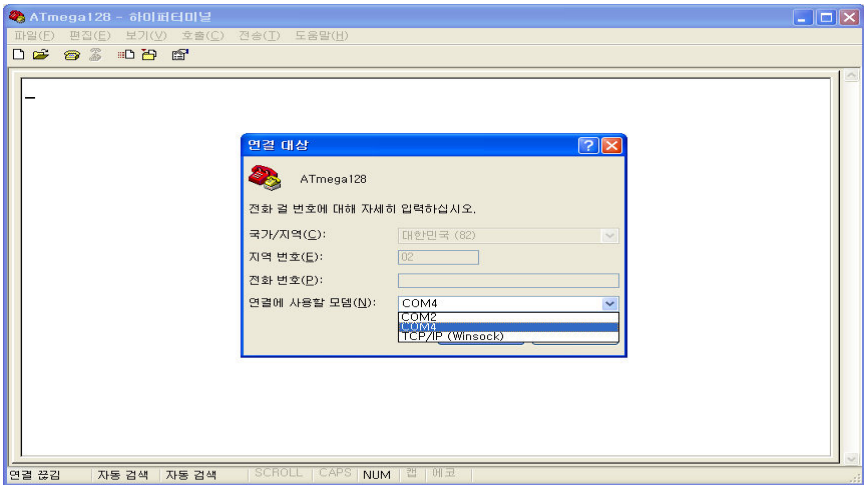
4. 하이퍼터미널 설정

실습 보드와 USART 통신을 하기 위해 Window에 기본적으로 설치되어 있는 하이퍼터미널 프로그램을 사용하기로 한다. 하이퍼터미널은 시작 메뉴에서 [보조 프로그램]-> [통신]-> [하이퍼터미널]을 실행하면 된다. 실행하고 나면 연결 설정을 물어보는 아래와 같은 창이 나타난다.

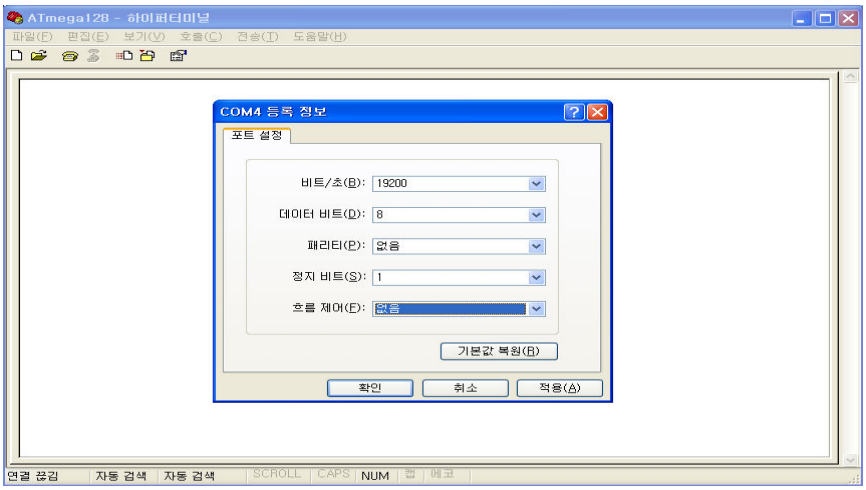


적당한 이름을 입력하고 '확인' 버튼을 클릭하면 된다. 여기서는 'ATmega128' 이라 입력한다.

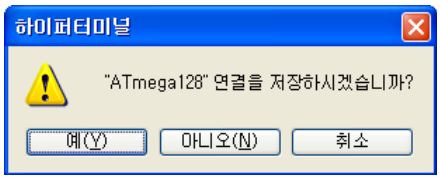
확인을 누르고 나면 '연결 대상' 이라는 창이 나타나는데 각자의 PC 환경에 맞는 포트를 선택하면 된다.



선택 후 '확인' 버튼을 누르면 COM 포트의 세부 설정을 위한 창이 나타난다. 실습 보드에서 설정한 값과 동일하게 설정해야 하므로 여기서는 19200bps, No Parity, 1bit Stop Bit, 8bit Data로 설정한다. '확인' 버튼을 클릭하면 모든 설정이 끝나게 된다.



하이퍼터미널을 사용 후 종료하게 되면 아래와 같은 창이 나타나는데, 여기서 '저장' 버튼을 클릭하여 저장하게 되면 다음 실행 시 설정 없이 현재 설정 값 그대로 불러와 사용할 수 있다.



저장된 연결은 시작 메뉴의 [보조 프로그램]-> [통신]-> [하이퍼터미널]-> [ATmega128]을 선택하여 불러올 수 있다.

실습 1. USART 송신하기

하이퍼터미널에 "Hello ATmega128!" 이라는 문구를 출력하라.

▣ USART_Tx.c (Project name - USART_Tx)

```
#include <avr/io.h>
#include <avr/delay.h>  /* _delay_ms() 함수 사용을 위한 헤더 파일 추가 */

/* =============== USART 제어를 위한 정의 ================ */
/* 1. UBRR에 저장할 Baud Rate 값을 정의 */
/*   여기서는 ATmega128 데이터 시트에 나와 있는 표를 참조하여 51(19200bps) 값을 정의 */
/*   BAUD_RATE = (16000000 / (16 * 19200)) - 1  */
#define   BAUD_RATE        51
/* ================================================== */

/* ============= USART 초기화 함수 구현 ============= */
/*  USART0 초기화 함수 정의 */
void USART_Init()
{
        /* 정의된 Baud Rate 값을 UBRR0H, UBRR0L에 설정 */
        UBRR0H = (unsigned char) BAUD_RATE >> 8;
        UBRR0L = (unsigned char) BAUD_RATE;

        UCSR0B = (1<<TXEN);                        /* Transmitter Enable
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);          /* No Parity, 1 Stop Bit, 8 bit Data
}
/* ================================================== */

/* ============= USART Transmitter 함수 구현 ============= */
/* USART 송신 함수 정의 */
void USART_Transmitter(unsigned char txData)
{
        while(!(UCSR0A & (1<<UDRE))) ;      /* 송신 버퍼가 비워질 때까지 대기 */
        UDR0 = txData;                      /* 버퍼에 데이터를 채운다. */
}
/* ================================================== */

/* ================== main 함수 구현 ================== */
int main(void)
{
        unsigned char buffer[] = "Hello ATmega128!";
        int i;
```

```
        USART_Init();       /* USART0 초기화 */
        while(1)
        {
                for(i=0; i<sizeof(buffer); i++)  /* 1바이트 씩 송신 */
                {
                        if(!buffer[i])
                                break;
                        USART_Transmitter(buffer[i]);
                }
                _delay_ms(1000);
        }
        return 0;
}
/* ================================================== */
```

시리얼 케이블로 실습 보드와 PC를 연결한 후 하이퍼터미널을 실행하고 실습 보드의 전원을 켜거나 Reset 시키면 하이퍼터미널 화면에 "Hello ATmega128!" 이라는 문구가 출력되는 것을 확인할 수 있다.

실습2. USART 수신하기

키보드로 입력하는 알파벳을 USART 통신으로 수신하여 하이퍼터미널에 출력하라.

▣ USART_Rx.c (Project name - USART_Rx)

```
#include <avr/io.h>
#include <avr/interrupt.h>

/* ================= USART 제어를 위한 정의 ================== */
/* 1. UBRR에 저장할 Baud Rate 값을 정의한다.*/
/*   여기서는 ATmega128 데이터 시트에 나와 있는 표를 참조하여 51(19200bps) 값을 정의한다.*/
/* #define BAUD_RATE        (16000000 / (16 * 19200)) - 1
#define   BAUD_RATE        51
/* ===================================================== */

/* ============= USART 제어를 위한 전역 변수 선언 ============= */
volatile unsigned char rxData;
/* ===================================================== */

/* ================ USART 초기화 함수 구현 ================= */
void USART_Init()
{
        /* 정의된 Baud Rate 값을 UBRR0H, UBRR0L에 설정한다. */
        UBRR0H = (unsigned char) BAUD_RATE >> 8;
        UBRR0L = (unsigned char) BAUD_RATE;

        /* Receive Complete Interrupt, Transmitter, Receiver Enable */
        UCSR0B = (1<<RXCIE) | (1<<TXEN) | (1<<RXEN);

        /* No Parity, 1 Stop Bit, 8 bit Data */
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);
}
/* ===================================================== */

/* ================= USART 송신 함수 구현 ================== */
void USART_Transmitter(unsigned char txData)
{
        while(!(UCSR0A & (1<<UDRE))) ;      /* 송신 버퍼가 비워질 때까지 기다린다. */
        UDR0 = txData;                      /* 버퍼에 데이터를 채운다. */
}
/* ===================================================== */
```

```
/* =============== USART Interrupt 함수 구현 ================ */
SIGNAL(SIG_USART0_RECV)
{
        /* 수신 버퍼에 읽지 않은 데이터가 있으면 값을 읽어온다. */
        /* 읽어 온 데이터를 다시 송신하여 하이퍼터미널에 출력되도록 한다. */
        if(UCSR0A & (1<<RXC))
        {
                rxData = UDR0;
                USART_Transmitter(rxData);
        }
}
/* ===================================================== */

/* ==================== Main 함수 구현 ===================== */
int main(void)
{
        USART_Init();           /* USART0 초기화 함수를 호출한다. */

        sei();                  /* 글로벌 인터럽트를 활성화 시킨다. */

        while(1)
        {
                ;               /* 인터럽트를 기다리는 Dummy Loop */
        }
        return 0;
}
/* ===================================================== */
```

*** 참고사항 - printf() 사용하기**

USART 통신을 사용하여 하이퍼터미널에 값을 출력할 수 있지만, 1바이트씩만 전송이 가능하기 때문에 숫자 또는 문자열을 출력하려면 상당히 번거로운 과정을 거쳐야 한다. 따라서 표준 출력 함수인 printf() 함수를 USART 통신으로 사용하도록 설정해보자. '실습 1' 의 예제 소스를 다음과 같이 수정한다.

▣ USART_Tx_printf.c (Project name - USART_Tx_printf)

```
#include <avr/io.h>
#include <avr/delay.h>
#include <stdio.h>              /* 표준 입출력을 위한 헤더파일 포함 */

/* ================= USART 제어를 위한 정의 ================== */
#define   BAUD_RATE        51
/* ===================================================== */

/* ============ USART 표준 출력 제어를 위한 정의 =============== */
static int USART0_putchar(char txData, FILE *stream);
static FILE device=FDEV_SETUP_STREAM(USART0_putchar, NULL, _FDEV_SETUP_WRITE);
/* ===================================================== */

/* =============== USART 표준 출력 함수 구현 ================ */
/* 표준 출력으로 사용할 USART0_putchar 함수 구현 */
static int USART0_putchar(char txData, FILE *steam)
{
        /* New Line 출력의 경우 '\n\r' 로 바꿔줘야 함 (\n : new line, \r : carriage return) */
        if(txData == '\n')
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\n';
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\r';
        }
        else
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = txData;
        }
        return 0;
}
/* ===================================================== */
```

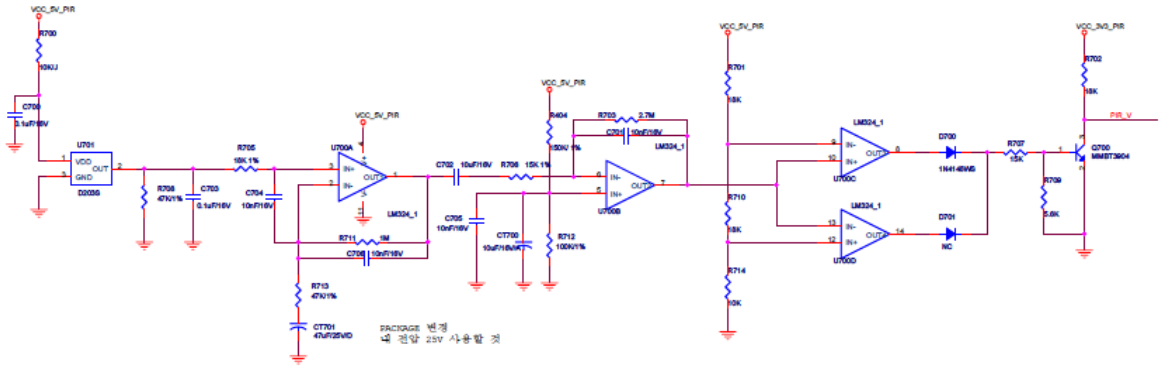
```
/* ================= USART 초기화 함수 구현 ================== */
void USART_Init()
{
        /* 정의된 Baud Rate 값을 UBRR0H, UBRR0L에 설정 */
        UBRR0H = (unsigned char) BAUD_RATE >> 8;
        UBRR0L = (unsigned char) BAUD_RATE;

        UCSR0B = (1<<TXEN);                 /* Transmitter Enable */
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);   /* No Parity, 1 Stop Bit, 8 bit Data  */
}
/* ====================================================== */

/* ================= USART 송신 함수 구현 ================== */
/* USART0_putchar 함수에서 출력되므로 송신함수를 따로 구현할 필요 없다. */
void USART_Transmitter(unsigned char txData)
{
        while(!(UCSR0A & (1<<UDRE))) ;      /* 송신 버퍼가 비워질 때까지 대기 */
        UDR0 = txData;                      /* 버퍼에 데이터를 채운다. */
}
/* ====================================================== */

/* ===================== Main 함수 구현 ==================== */
int main(void)
{
        /* unsigned char buffer[] = "Hello ATmega128!"; */
        int i;

        USART_Init();               /* USART0 초기화 */
        stdout = &device;           /* 표준출력으로 사용할 장치 설정 */

        /* printf() 사용하여 출력 */
        while(1)
        {
                printf("Hello ATmega128!\n");
                _delay_ms(1000);
        }
        return 0;
}
/* ====================================================== */
```

실습 5. PIR 센서 제어하기

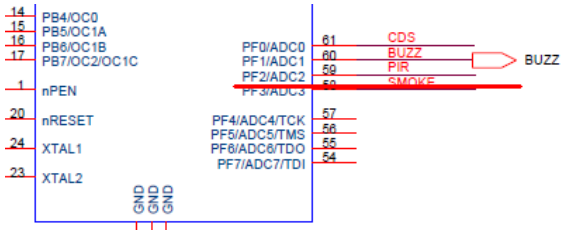
5.1 PIR 센서 이해하기

PIR 센서는 적외선을 이용하여 인체 등에서 발생하는 소량의 열을 감지하는 센서로 인체 감지 센서라고도 하며, 정확히 말하면 일정한 적외선을 띈 물체가 움직이는 것을 감지하는 센서이다. 열의 움직임을 감지하기 때문에 센서를 향해 똑바로 직선으로 움직이는 것은 감지하지 못한다는 단점이 있어, 주로 물결무늬 등으로 열을 일부 분산시키는 덮개와 함께 사용한다.

적외선을 받아들여 열의 움직임이 감지되지 않을 때는 Output Pin으로 'Low' 신호를 출력하다가, 움직임이 감지되면 'High' 신호를 출력한다. 다음은 PIR 센서의 회로도를 나타낸 것이다.



실습 보드에서 PIR 센서의 Output Pin은 다음과 같이 연결되어 있다.



위의 그림에서 볼 수 있듯이 PIR 핀은 PortF의 2번째 핀에 연결되어 있으므로, PORTF의 2번째 비트를 읽어 값을 측정할 수 있다. 비교적 간단한 실습이므로 다음 예제 소스를 따라하면서 직접 응용해보도록 한다.

5.2 PIR 센서 제어 실습하기

실습 1. PIR 센서 제어 실습하기

PIR 센서로부터 값을 받아 열이 감지되면 LED가 켜지도록 소스를 작성하라.

▣ PIR_Ctrl.c (Project name - PIR_Ctrl)

```
#include <avr/io.h>

/* ================= USART 제어를 위한 정의 ================== */
#define   DDR_LED          DDRE
#define   PORT_LED         PORTE
#define   LED1_BIT         5
/* ======================================================== */

/* ================= PIR 센서 제어를 위한 정의 ================= */
/* PIR 센서 제어를 위한 Port 정의 */
#define   DDR_PIR          DDRF
#define   PIN_PIR          PINF
#define   PIR_BIT          2
/* ======================================================== */

/* ====================== Main 함수 구현 ==================== */
int main()
{
        DDR_LED |= (1<<LED1_BIT);  /* LED Port Set Output Mode  */
        DDR_PIR &= ~(1<<PIR_BIT);  /* PIR Port Set Input Mode  */

        while(1)
        {
                if(PIN_PIR & (1<<PIR_BIT))
                        PORT_LED |= (1<<LED1_BIT);        /* LED On */
                else
                        PORT_LED &= ~(1<<LED1_BIT);       /* LED Off */
        }
        return 0;
}
/* ======================================================== */
```

실습 6. DC 모터 제어하기

6.1 DC 모터 이해하기

모터(전동기)는 기기가 어떤 일을 할 때 그 기기의 구동력이 되는 것이다. 모터에 전원을 넣으면 전기 작용에서 발생되는 회전력을 꺼내어 다른 기계에 힘을 전달시키는 것으로 전기 에너지를 기계 에너지(회전 에너지)로 변환하는 장치이다.

모터는 사용하는 전류를 기준으로 교류전류(AC : Alternating Current)를 사용하는 AC 모터와 직류전류(DC : Direct Current)를 사용하는 DC 모터로 분류한다.

직류는 +/- 극성이 존재하는 것으로 예를 들어서 시계에 건전지를 넣을 경우 정해진 방향으로 건전지를 넣지 않으면 시계가 동작하지 않는다. 전류가 흐르는 방향이 정해져 있기 때문이다.

반면 교류는 가전제품을 사용하기 위해서 플러그에 꽂을 때 꽂는 방향에 상관없이 동작하는 것을 보았을 것이다. 도선을 따라서 흐르는 전류의 방향이 계속 바뀌어서 흐르기 때문에 방향에 상관없이 꽂아도 동작을 하는 것이다.

DC 모터의 특징은 아래와 같다.

- 장점 - 전류의 세기가 고르기 때문에 전기 제품에 손상을 주지 않음
- 단점 - 구조가 복잡하고 전압을 높이기가 쉽지 않아 장거리 송전이 어렵고 교류로 변환이 어려움
- 용도 - 소형 전자제품에 사용

AC 모터의 특징은 아래와 같다.

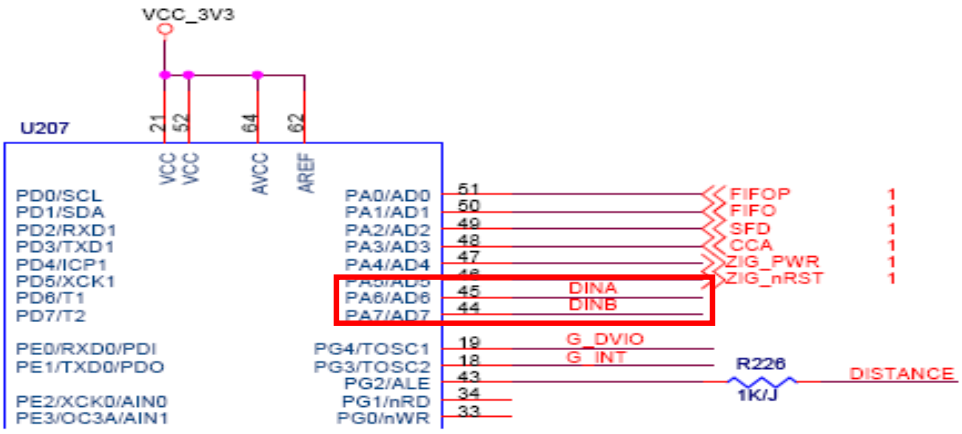
- 장점 - 구조가 간단하며 전압을 쉽게 높일 수 있어서 장거리 송전 가능, 직류로 변환이 쉬움
- 단점 - 전압이 고르지 않아 전기 회로에 손상을 줄 수도 있음
- 용도 - 대부분의 전자제품에 사용

보다 자세한 사항을 다른 참고 서적을 참고해서 살펴보기 바란다.

6.2 DC 모터 제어 실습하기

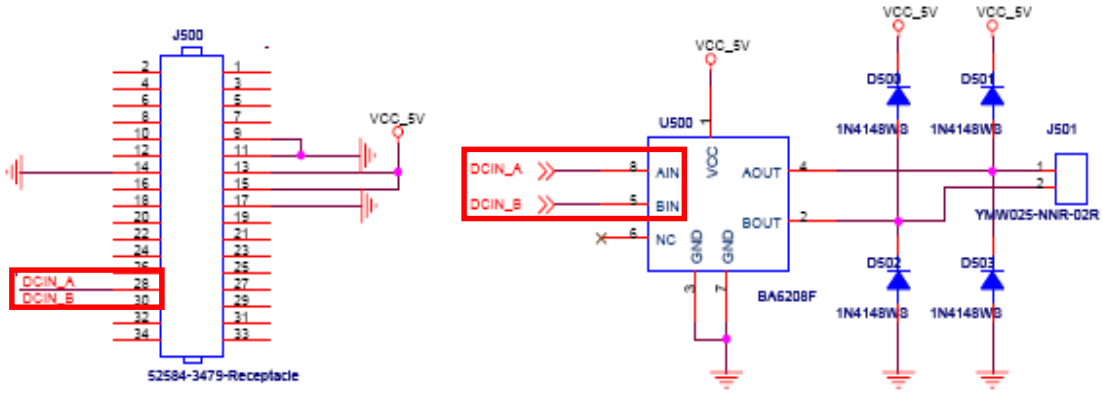
우리 실습 보드에서는 GPIO 포트를 이용하여 DC 모터를 제어한다.

DC 모터를 제어하기 위해서는 실습 보드의 DC 모터가 ATmega128 칩의 어느 핀에 연결되어 있는지 회로도 상에서 확인해야 한다. 지금 우리가 사용하고 있는 실습 보드의 회로도를 살펴보면 DC 모터는 포트 A의 6~7번 핀에 연결되어 있다.



DC 모터 회로도를 살펴보면 DC 모터를 구동시키기 위해서는 I/O 포트에 1(High)을 설정해야 한다.

DDRx가 출력으로 설정되어야 하며, PORTx에 설정하는 값에 따라 DC 모터 구동 여부 및 방향이 결정된다.



PORTA 7 핀이 1(High)이면 시계방향(정방향)으로 모터가 회전하고 PORTA 6 핀이 1(High)이면 반 시계방향(역방향)으로 모터가 회전한다. 그리고 PORTA 6, 7 핀이 모두 1(High)이면 모터가 정지한다.

실습 보드에서 DC 모터가 포트 A의 6~7번에 연결되어 있으므로 DDRA와 PORTA를 사용하여 DC 모터를 제어할 수 있다.

실습 1. DC 모터 구동하기

▣ DCMotor_Ctrl.c (Project Name - DCMotor_Ctrl)

```
#include <avr/io.h>

/* ================== DC 모터 제어를 위한 정의================= */
#define DCM_DDR             DDRA
#define DCM_PORT            PORTA
/* ================================================== */

/* ================== LED 제어를 위한 정의================== */
#define LED_DDR   DDRE
#define LED_PORT  PORTE
/* ========================================================== */

/* ================= LED 켜기 함수 구현 ====================== */
void setLED(unsigned int bit_num)
{
        if(bit_num == 0)
        {
                LED_DDR         = 0xE0;     /* Output으로 설정 */
                LED_PORT        = 0xE0;     /* OFF : 0  / ON : 1 */
        }else{
                LED_PORT        = (1<<(bit_num));
        }
}
/* ================================================== */

/* ================= DC 모터 초기화 함수 구현 ================= */
void initDCM(void)
{
        DCM_DDR  |=  (1<<7)|(1<<6);             /* PA6, PA7 비트 Output으로 설정 */
        DCM_PORT &= ~((1<<7) | (1<<6));        /* 7, 6번 bit만 Clear */
        DCM_PORT |=  (1<<7)|(0<<6);             /* 정방향 7, 6번 bit만 set */

        setLED(0);
}
/* ================================================== */

/* ====================== Main 함수 구현 ================== */
int main(void)
```

```
{
        initDCM();
        return 0;
}
/* ========================================================== */
```

실습 2. DC 모터를 정방향/역방향/정지 하도록 구현하기

▣ DCMotor2.c (Project Name - DCMotor2)

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

/* ==================== DC 모터 제어를 위한 정의================== */
#define DCM_DDR   DDRA
#define DCM_PORT PORTA
/* ========================================================== */

/* ==================== LED 제어를 위한 정의==================== */
#define LED_DDR   DDRE
#define LED_PORT  PORTE
/* ============================================================== */

/* ================= LED 켜기 함수 구현 ====================== */
void setLED(unsigned int bit_num)
{
        if(bit_num == 0)
        {
                LED_DDR  = 0xE0;    /* Output으로 설정  */
                LED_PORT = 0xE0;    /* OFF : 0  / ON : 1 */
        }
        else
        {
                LED_PORT = (1<<(bit_num));
        }
}
/* ========================================================== */

/* ================= 정방향 회전 함수 구현 ==================== */
void forward_DCM()
{
```

```
        DCM_PORT &= ~((1<<7) | (1<<6));  /* 7, 6번 bit만 Clear */
        DCM_PORT |=  (1<<7)|(0<<6);
}
/* ========================================================== */


/* ================= 역방향 회전 함수 구현 ==================== */
void backward_DCM()
{
        DCM_PORT &= ~((1<<7) | (1<<6));  /* 7, 6번 bit만 Clear */
        DCM_PORT |=  (0<<7)|(1<<6);
}
/* ========================================================== */

/* ================== 정지 함수 구현 ======================== */
void stop_DCM()
{
        DCM_PORT =  (1<<7)|(1<<6);
}
/* ========================================================== */

/* ================= 방향/LED 전환 함수 구현 ================== */
void runDC()
{
        while(1)
        {
                _delay_ms(1000);
                setLED(5);
                forward_DCM();   /* 시계방향(정방향) */

                _delay_ms(2000);
                setLED(6);
                backward_DCM();  /* 시계 반대방향(역방향)*/

                _delay_ms(2000);
                setLED(7);
                stop_DCM();          /* DC 모터 구동 정지 */
        }
}
/* ========================================================== */
```

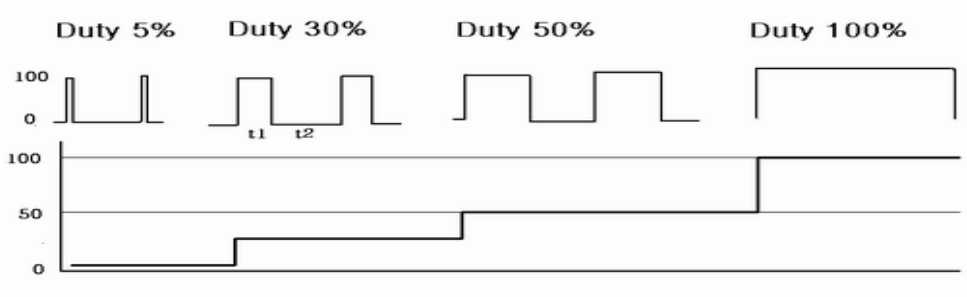
```
/* ================= DC 모터 초기화 함수 구현 ================== */
void initDCM(void)
{
        DCM_DDR  |=  (1<<7)|(1<<6);     /* PA6, PA7 비트 Output으로 설정 */
        DCM_PORT |=  (1<<7)|(1<<6);     /* DC 모터 구동 정지로 설정 */

        setLED(0);
}
/* ======================================================== */

/* ==================== Main 함수 구현 ===================== */
int main(void)
{
        initDCM();    /* DCM 초기화 */
         runDC();      /* DC 모터 작동 - 시계방향, 반시계 방향 */

        return 0;
}
/* ======================================================== */
```

실습 7. Step 모터 제어하기

7.1 PWM(Pulse Width Modulation) 이해하기

PWM이란 Pulse Width Modulation 약자로 펄스폭을 조절해서 원하는 값을 만들어 내는 기술이다.
디지털 신호에 변경을 가해서 아날로그 신호를 표현하는 방법으로써 일정 크기의 DC 전압이 공급되면 그 전압을 On/Off해서 pulse의 폭에 변화를 주는 것이다.

디지털 값은 0과 1밖에 출력하지 못한다. 0은 아날로그 값으로는 0V가 되는 것이고, 1은 내보내는 전압을 의미한다. 그러면 디지털로는 0V, +5V 두 가지 밖에 출력이 안 된다.



100us 동안 10us 는 +5V를 출력하고, 90us 는 0V를 출력하게 하자. 그러면 실제로는 10us 동안은 +5V가 나오고 90us 동안은 0V가 나오지만 그 값을 받는 DC 모터 같은 경우는 10us, 90us 의 시간은 너무 짧은 간격이라 10us 동안만 가고, 90us동안은 선다는 개념이 없다. 게다가 내부에 있는 코일과 기타 여러 영향으로 일종의 LPF 같이 작동되면서 5V의 10%되는 값, 즉 0.5V 가 주어진 것으로 작동하게 된다.

이 비율을 적당히 조절해서 DC 모터 구동을 시키는 것이다. 천천히 돌리려면 10% 비율로, 중간 속도로 돌리려면 50% 비율로, 빨리 돌리려면 100%로 돌리면 된다.

Pulse에 On 시간/ Off 시간의 비를 duty비라 하면 위의 그림처럼 On/Off 시간 간격을 조절해서 원하는 변화를 줄 수 있다. On 시간이 길수록 가해지는 전압은 높아지므로 모터의 속도가 빨라진다.

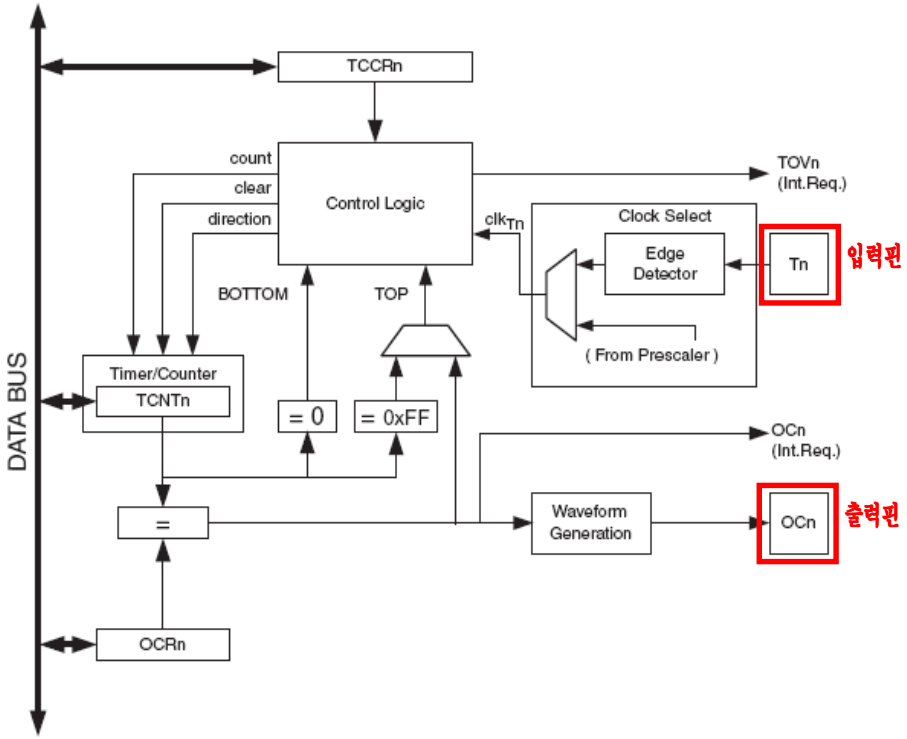
ATmega128에서 PWM은 Output Compare Math Interrupt 기능을 이용하여 TCNT와 OCR 값을 동작 중에 계속 비교하여 그 값이 같아지면 Output Compare Math Interrupt가 발생하며, OC(Output Compare) 핀에 신호가 출력되도록 한다. 이때 OCR의 값에 따라서 주기와 듀티비(duty ratio)를 가변 할 수 있는 출력신호를 발생하는 것이다.
모든 Timer/Counter의 PWM 제어의 기본 원리이며, 각 Timer/Counter별로 입/출력 핀과 레지스터 설정이 조금씩 다를 뿐이다. 좀 더 자세한 사항은 Timer/Counter2와 연관 지어서 살펴보도록 하자.

7.2 Timer/Counter2 이해하기

ATmega128의 Timer/Counter0~3에서는 PWM((Pulse Width Modulation : 펄스 변복조) 모드를 지원하고 있는데, 그 중에서 실습보드에 Step 모터와 연결되어 있는 Timer/Counter2를 통해서 PWM에 대해서 살펴보도록 한다.

Timer/Counter2는 Timer/Counter0과 함께 8비트 구조로 유사한 기능을 수행하지만, Timer/Counter0처럼 32.768kHz의 크리스탈을 접속하는 TOSC1 단자를 통해서 외부 클럭 인가가 불가능하다. Timer/Counter2는 T2 핀으로 입력되는 외부 클럭을 사용하는 카운터 모드 기능을 가지는 것을 빼고는 Timer/Counter0과 유사한 기능을 한다.

다음은 ATmega128의 8bit Timer/Counter2의 Block Diagram이다.



Timer/Counter2 기본동작은 다음과 같다.

· Timer/Counter2에서 사용할 clock (clk_{T2}) 선택
  - 시스템 클럭(clk_{IO}) 또는 T2 핀의 외부 입력 클럭 중 선택
  - TCCR2[CS22, CS21, CS20] 비트 설정으로 입력된 클럭을 prescaler하여 출력 clock 설정

Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T2 pin. Clock on falling edge
1	1	1	External clock source on T2 pin. Clock on rising edge

· TCNT2 동작 모드에 따른 제어

- TCCR2 (Timer/Counter2 Control Register)로 동작모드 설정

동작 모드 : TCCR2[WGM21, WGM20] 비트 설정으로 Overflower, CTC, PWM 설정

TCCR2[COM21, COM20] 비트 설정으로 모드별 CO2 핀 설정

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Toggle OC2 on compare match
1	0	Clear OC2 on compare match
1	1	Set OC2 on compare match

· Mode별 동작 제어

- clock 마다 TCNT2 값 변경

위상 변경 PWM 모드 : 증가/감소

그 외 : 증가

- TCNT2와 OCR0 일치 시 모드별 처리

OC0 핀 출력 설정된 경우 : 0C0핀으로 출력

비교 일치 인터럽트 요청

- TCNT2 값에 따른 처리

TCNT2 값이 BOTTOM(0x00) / TOP(OxFF) 일 경우 : Overflower 인터럽트 요청

Timer/Counter2도 다음과 같은 4개의 모드를 가진다.

각 모드별 특징을 간략하게 살펴본다.

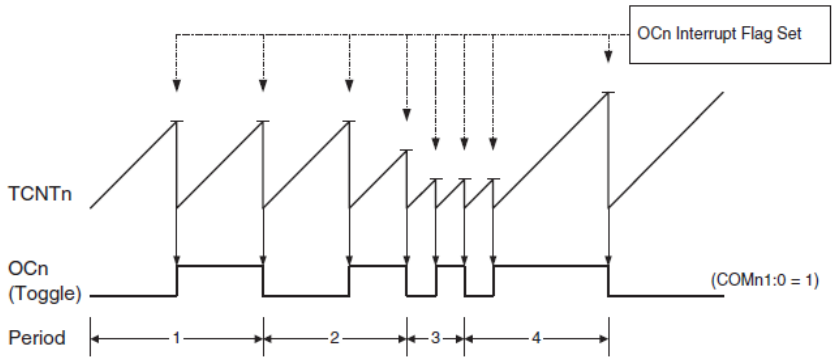
1. 타이머 모드 (Normal Mode)

일반적인 방법으로 클록이 있을 때마다 8bit 레지스터(TCNT2)의 값이 증가하고, 최대값(0xFF)이 되면 다시 '0'으로 Clear 되며, 이때 Overflow Flag(TIFR[TOV2])가 '1'이 되어 Overflow 인터럽트를 요청한다.

2. CTC 모드 (Clear Timer on Compare Match Mode)

클록이 있을 때마다 8bit 레지스터(TCNT2)의 값이 증가하고, 비교 레지스터(OCR2)의 값과 비교해서 일치하면 비교 일치 인터럽트를 요청한다. 이 때 8bit 레지스터(TCNT2)의 값은 '0'으로 Clear 된다.

다음은 Timer/Counter2의 CTC 모드 타이밍도이다.



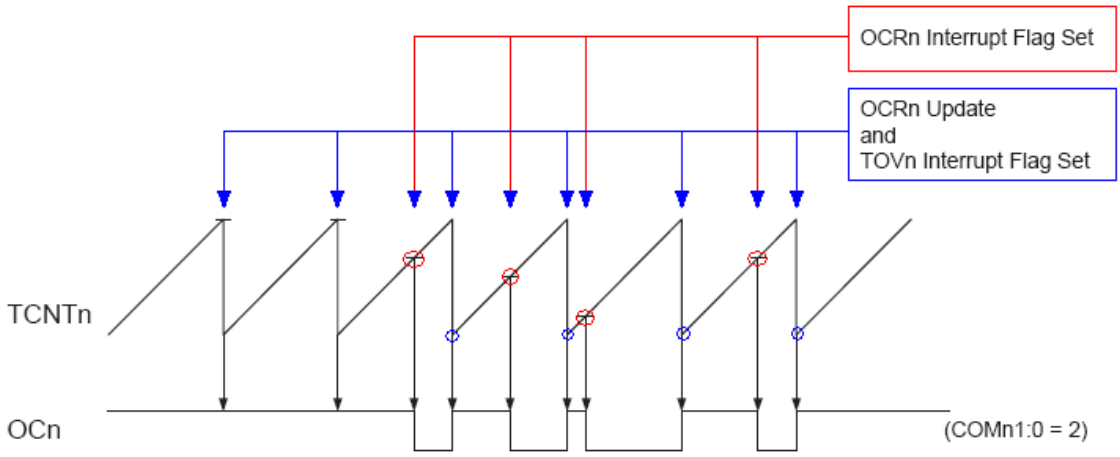
단, 비교 레지스터(OCR0)의 값을 변경할 때, TCNT2의 현재 값보다 작은 값을 써 넣으면 비교 일치 동작을 놓칠 수 있으므로 주의하여야 한다.

3. 고속 PWM 모드

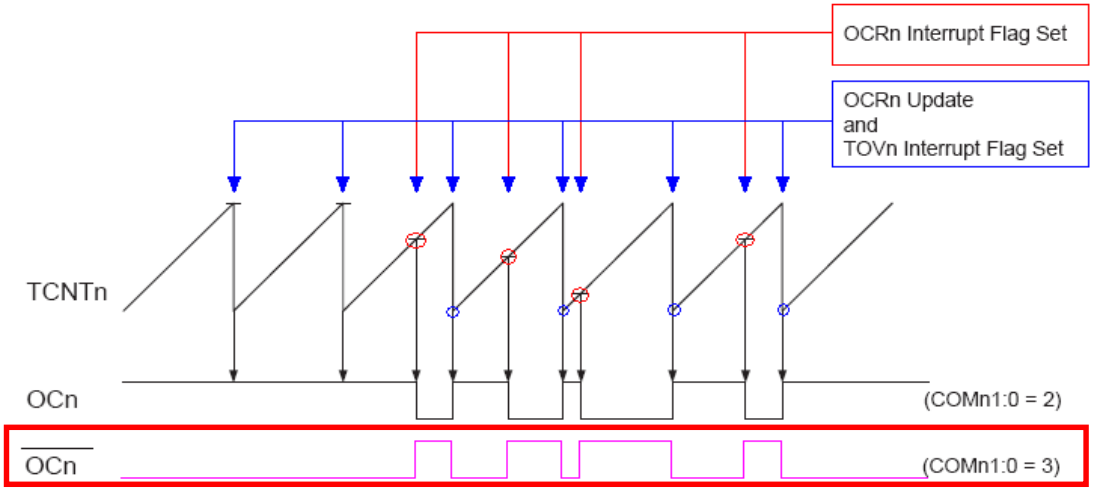
PWM은 Pulse Width Modulation의 약자로 높은 주파수의 PWM 출력 파형을 발생하는 모드 이다.

기본 동작 방법은 다음과 같다.

- Timer/Counter의 TCNT2는 0x00 ~ 0xFF 범위에서 증가 방향으로만 반복적으로 수행
- TCNT2의 값은 항상 OC2 값과 비교
  TCNT2와 OC2가 일치할 경우 OC2 출력신호에 0 (Low) 값이 출력
- TCNT2가 0xFF이 되면 Overflow 발생하고 CO2 출력신호가 1(High)로 set
  OCR2 갱신



위에 설명한 방식으로 동작하는 고속 PWM 모드와 반대로 동작하는 것을 인버팅 모드는 아래 그림과 같다.
기본 동작원리는 같고, OC2에 출력되는 신호가 반대로 출력이 된다.



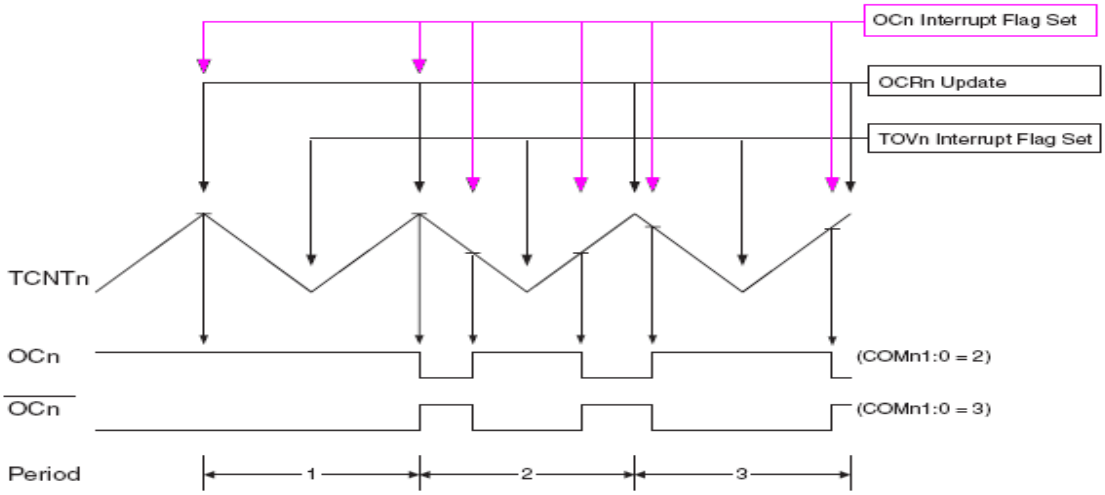
4. 위상 변경 PWM 모드

PC(Phase Correct) PWM은 PWM 출력신호의 약 1/2로 낮은 주파수가 출력된다.

PC(Phase Correct) PWM 모드에서는 OCR2가 이중 버퍼링 기능을 가지고 있어서, OCR2에 새로운 값을 라이트 하더라도 OCR2에 즉시 변경되지 않고, TCNT가 0xFF로 되는 순간에 갱신이 된다.

기본 동작 방법은 다음과 같다.

- Timer/Counter의 TCNT2는 0x00 ~ 0xFF ~ 0x00 범위에서 상향증가/하향감소 반복적으로 수행
- TCNT2의 값은 항상 OC2 값과 비교
  상향증가 시 일치 TCNT2와 OC2가 일치할 경우 OC2 출력신호에 0 (Low) 값이 출력
  하향감소 시 일치 TCNT2와 OC2가 일치할 경우 OC2 출력신호에 1 (High) 값이 출력
- TCNT2가 0xOO가 되면 TOV2 = 1로 set되고 Overflow Interrupt 요청
  이 인터럽트가 처리되기 시작하면 TOV2 = 0으로 set
  OC2 출력신호가 1(High)로 set
- TCNT2가 0xFF 되면 OCR2 값 갱신, TOV2 = 1로 set



7.3 Timer/Counter2 레지스터

활용 방법에 앞서 주로 사용하는 Timer/Counter 관련 레지스터들에 대해서 알아보자. 실습에서 자주 사용할 레지스터들에 대해 설명할 것이며, 자세한 내용은 데이터 시트를 참조한다.

1. TCCR2 (Timer/Counter2 Control Register)

Timer/Counter2의 Clock 선택 및 모드를 설정한다.

Bit	7	6	5	4	3	2	1	0	
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
Read/Write	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	

- **FOC2** (Force Output Compare) : WGM 비트가 PWM 모드가 아닐 경우 유효하며, Write만 가능하다.
  - '1' 을 써 넣는 즉시 파형 발생 Unit이 강제로 동작하여, COM21,20에 설정된 대로 OC2로 출력한다.
  - 이 비트로 인한 인터럽트는 발생되지 않으며, CTC 모드에서 OCR2가 TOP이 되어도 타이머는 Clear 되지 않는다.

- **WGM20, WGM21** (Waveform Generation Mode) : 파형 출력 기능을 설정하기 위한 비트이다.

Mode	WGM21 (CTC2)	WGM20 (PWM2)	Timer/Counter Mode of Operation	TOP	Update of OCR2 at	TOV2 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

  - Normal Mode는 Overflow 인터럽트를 사용하는 기본적인 모드이다.
  - 비교 인터럽트를 사용하기 위해서는 CTC Mode로 설정한다.
    CTC Mode로 설정했을 경우 위의 표와 같이 TOP이 0xFF가 아닌 OCR0 값이 된다.
  - 위상변경 PWM Mode로 설정한 경우 TOP이 OxFF가 되며, TCNT2가 TOP이 될 경우 OCR2값이 변경된다.

- **COM21, COM20** (Compare Match Output Mode) : OC2의 출력을 제어한다. WGM21,20에 따라 결정된다.
  - PWM Mode가 아닐 경우

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Toggle OC2 on compare match
1	0	Clear OC2 on compare match
1	1	Set OC2 on compare match

00 : 포트로 동작하며, OC2은 사용하지 않는다.
01 : TCNT2과 OCR2의 값이 일치하면 OC2 값을 toggle 시킨다.
10 : TCNT2과 OCR2의 값이 일치하면 OC2 값을 '0' 으로 Clear 한다.
11 : TCNT2과 OCR2의 값이 일치하면 OC2 값을 '1' 로 Set 한다.

• Fast PWM Mode 일 경우

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match, set OC2 at BOTTOM, (non-inverting mode)
1	1	Set OC2 on compare match, clear OC2 at BOTTOM, (inverting mode)

00 : 포트로 동작하며, OC2는 사용하지 않는다.
01 : 사용안함
10 : 처음 OC2 핀 'H', TCNT2가 증가해서 OCR2 = TCNT2으면 OC2 핀 'L'로 Set
TCNT2가 계속 증가해서 TOP이 되면 CO2핀 'H' 출력
11 : 처음 OC2 핀 'L', TCNT2가 증가해서 OCR2 = TCNT2으면 OC2 핀 'H'로 Set
TCNT2가 계속 증가해서 TOP이 되면 CO2핀 'L 출력

• Phase Correct PWM Mode 일 경우

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match when up-counting. Set OC2 on compare match when downcounting.
1	1	Set OC2 on compare match when up-counting. Clear OC2 on compare match when downcounting.

00 : 포트로 동작하며, OC2는 사용하지 않는다.
01 : 사용안함
10 : 처음 OC2 핀 'H', TCNT2가 증가해서 OCR2 = TCNT2으면 OC2 핀 'L'로 Set
TCNT2가 계속 증가해서 TCNT2가 0xFF가 되어 다시 0x00이 되면 OC2 핀 'H'
11 : 처음 OC2 핀 'L', TCNT2가 증가해서 OCR2 = TCNT2으면 OC2 핀 'H'로 Set
TCNT2가 계속 증가해서 TCNT2가 0xFF가 되어 다시 0x00이 되면 OC2 핀 'L'

- **CS22, CS21, CS20** (Clock Select) : 사용할 클록과 분주비(Prescaler)를 선택한다.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T0S}/(No prescaling)
0	1	0	clk_{T0S}/8 (From prescaler)
0	1	1	clk_{T0S}/32 (From prescaler)
1	0	0	clk_{T0S}/64 (From prescaler)
1	0	1	clk_{T0S}/128 (From prescaler)
1	1	0	clk_{T0S}/256 (From prescaler)
1	1	1	clk_{T0S}/1024 (From prescaler)

2. TCNT2 (Timer/Counter Register)

Bit	7	6	5	4	3	2	1	0	
	TCNT2[7:0]								TCNT2
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	

하드웨어에 의해 자동으로 증가하는 8 bit 카운터 값이다. 카운터가 동작하는 동안 TCNT2 값을 변경하는 것은 TCNT2과 OCNT2의 비교에 실패할 위험이 있다.

3. OCR2 (Output Compare Register)

Bit	7	6	5	4	3	2	1	0	
	OCR2[7:0]								OCR2
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	

카운터 값인 TCNT2과 비교가 되는 8 bit 값이다. 클록마다 비교하여 비교 인터럽트나 OC2 핀으로 파형을 내보내기 위해 사용된다.

4. TIMSK (Timer/Counter Interrupt Mask Register)

Timer/Counter의 인터럽트 요청을 제어하며, Timer/Counter2에서 사용하는 기능만 살펴본다.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

- OCIE2 (Timer/Counter2 Output Compare Match Interrupt Enable) : 비교 일치 인터럽트를 허용한다. 이 비트가 '1' 로 되어 있고, 상태 레지스터(Status Register)의 I 비트가 '1' 로 되어 있으면 비교 일치 인터럽트가 활성화된다.
- TOIE2 (Timer/Counter2 Overflow Interrupt Enable) : Overflow 인터럽트를 허용한다. 이 비트가 '1' 이고, 상태 레지스터(Status Register)의 I 비트가 '1' 로 설정되면 Overflow 인터럽트가 활성화된다.

5. TIFR (Timer/Counter Interrupt Flag Register)

Timer/Counter0 ~ 2가 발생하는 인터럽트 플래그를 저장하는 레지스터이다.

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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	

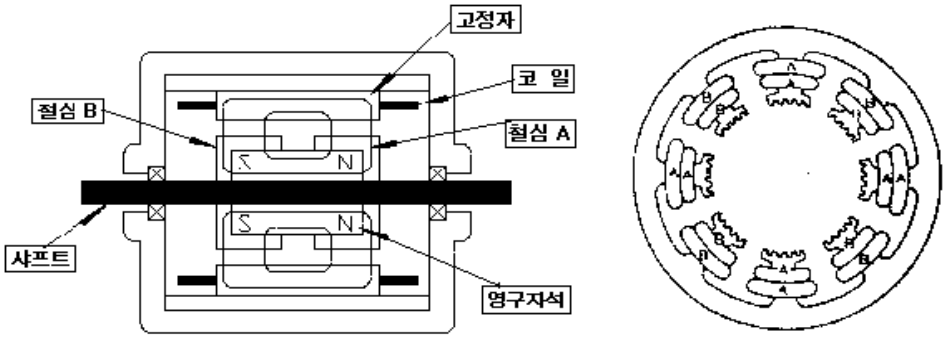
- OCF2 (Output Compare Match Flag 2) : Timer/Counter2의 TCNT2와 OCR2 값이 비교/일치하면 1로 set하고 Output Compare Match Interrupt가 요청된다. 이 인터럽트가 처리되기 시작하면 자동적으로 0 으로 clear된다.
- TOF2 (Timer/Counter2 Overflow Flag) : Timer/Counter2에서 Overflow 발생하면 1로 set하고 Overflow Interrupt가 요청된다. 이 인터럽트가 처리되기 시작하면 자동적으로 0 으로 clear된다.

7.4 Step 모터 이해하기

펄스에 의해 일정한 각도씩 제어할 수 있는 모터로 Step motor, Pulse motor, Stepper motor 라고도 하며 한 걸음 한 걸음 단계적으로 움직이는 모터이다. 주로 위치 결정 용도로 사용된다.

pulse에 의해 디지털적으로 제어하기 적합한 모터로 micom에서 사용하기에 적합한 모터이다.
외부의 DC 전압 또는 전류를 모터의 각 상 단자에 스위칭방식으로 입력시켜 줌에 따라 일정한 각도의 회전을 한다. 따라서 이는 일종의 디지털 제어방식의 기기로서 일반적인 아날로그 전원보다는 디지털 펄스형식의 제어에 적합하다.
즉, 디지털 펄스 1개에 1개 스텝에 해당하는 회전각만큼 정확한 회전운동을 하게 되며, 입력펄스의 수와 단위시간당 펄스입력속도에 정확히 비례하여 연속운동을 하게 된다 (단, 이때의 연속운동이란 불연속운동이 펄스입력속도가 증가함에 따라 마치 연속운동의 효과를 내는 것처럼 보일뿐이다.

내부를 구성하는 고정자라 불리는 극의 수에 따라 단상, 2상, 3상, 4상, 5상, 6상 등의 정류가 있으며, 이 극의 수에 따라 motor의 step 각 등의 기본 특징이 달라진다.



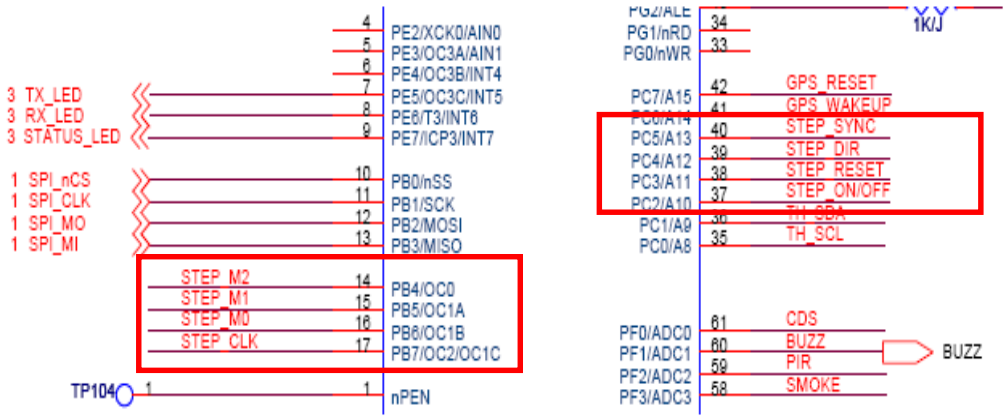
일정한 각도씩 회전 또는 직선 운동을 하는 디지털 액추에이터라 할 수 있다. 일반적으로 회전 운동을 하는 경우 스텝 모터라 부르고, 직선 운동을 하는 경우에는 리니어 스텝 모터라 부른다.

스텝 모터의 특징은 다음과 같다.
- 회전 각도는 입력 펄스 신호수에 비례하여 정해진다.
- 회전속도는 입력 펄스 레이트(펄스 주파수)에 비례한다.
- 회전자에 영구 자석을 사용하면 무자력 상태에서도, 자기보지력(self-holding torque, detent torque)가 발생한다.
- 고토크, 고속 응답, 소형 경량이다.
- 미소각, 고정도, 저가격이다.
- DC 모터의 브러쉬에 의한 기계적인 마찰로 인한 파손이 없기 때문에, 보수가 필요 없다.

Step 모터를 제어하기 위해서는 실습 보드의 Step 모터가 ATmega128 칩의 어느 핀에 연결되어 있는지 회로도 상에서 확인해야 한다. 지금 우리가 사용하고 있는 실습 보드의 회로도를 살펴보면 Step 모터는 포트 B의 4~7번 핀과 포트 C의 2~5번 핀에 연결되어 있다.

우선 Step 모터 관련 각 핀에 대해서 살펴보자. 아래 그림에서 보면 8개의 핀이 Step 모터 동작에 사용된다. 각 핀의 역할을 살펴보자.

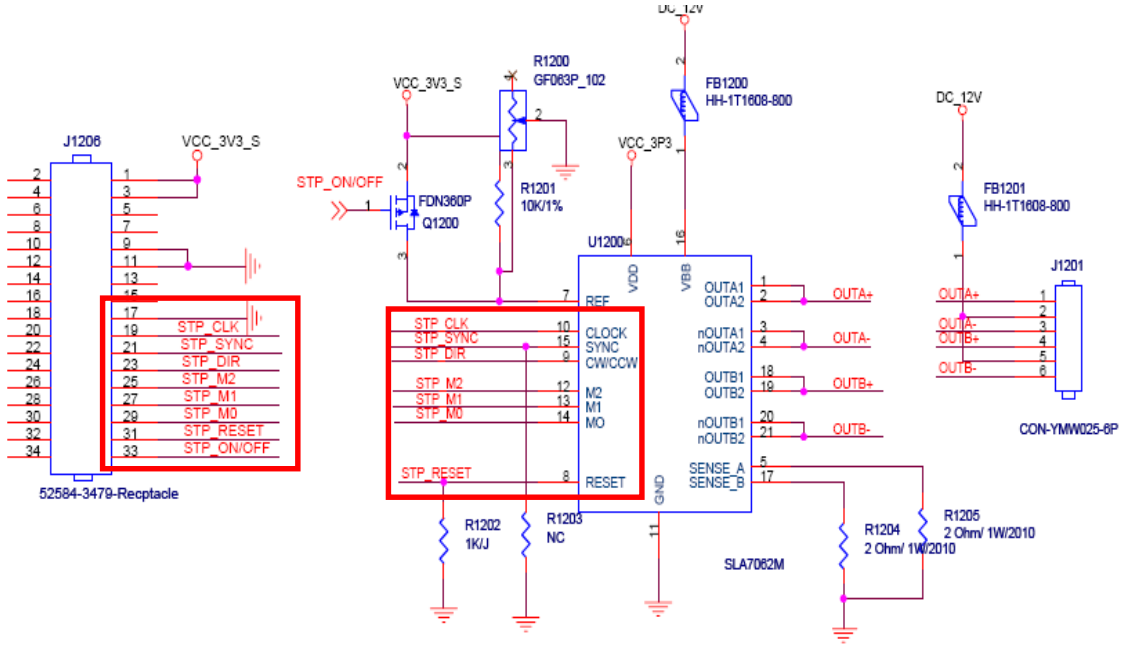
- STEP_ON/OFF : Step 모터의 동작 여부 설정
- STEP_DIR : Step 모터의 회전 방향
- STEP_CLK : Step 모터에 클럭 인가
- STEP_M1~M2 : Step 모터의 회전 각도 설정



Step 모터를 구동시키기 위해서는 우선 STEP_ON/OFF 기능의 PortC 2번 핀에 1(High)을 설정을 하고, STEP_CLK 기능의 PortB 7번 핀에 Clock을 인가해야 한다. PortB 7번 핀에는 PWM의 주파수를 인가해서 Step 모터의 속도를 조절한다.

그리고, STEP_DIR 기능의 PortC 4번 핀의 설정으로 시계/반시계 방향의 회전 방향을 설정하고, STEP_M1~M2 기능의 PortB 5,6번 핀의 설정으로 회전 각도를 제어한다.

아래 회로도는 실습보드와 Step 모터와 연결되는 핀 회로도 이다.



7.5 Step 모터 제어 실습

실습 1. PWM 이용한 Step 모터 제어하기

▣ PWM_Step1.c (Project Name - PWM_Step01)

```
#include <avr/io.h>
#include <avr/interrupt.h>                    /* Interrupt 관련 Header file */

/* ================ PWM/Step 모터 제어를 위한 정의 ================ */
#define PWM_OC_DDR          DDRB        /* PB7 : OC2 Clcok Output Signal */
#define PWM_OC_PORT         PORTB

#define STEP_MOTOR_DDR_01   DDRB      /*  PB4(M2), PB5(M1), PB6(M0), PB7(CLK) */
#define STEP_MOTOR_PORT_01  PORTB

#define STEP_MOTOR_DDR_02   DDRC      /*  PC4 (DIR), PC2 (ON/OFF) */
#define STEP_MOTOR_PORT_02  PORTC
/* ============================================================ */

/* ===================== PWM 제어를 위한 정의===================== */
unsigned int gDutyValue = 50;    /* 듀티비를 설정하는 전역변수 선언 */
/* ============================================================ */

/* ==================== PWM/Step 함수 선언 ===================== */
void initPWM();
void init_StepMotor();
/* ============================================================ */

/* ======================== Main 함수 구현 ======================== */
int main()
{
        init_PWM();             /* PWM 관련 port 초기화*/
        init_StepMotor();       /* Step Motor 관련 port 초기화*/
        return 0;
}
/* ============================================================ */

/* ==================== PWM 초기화 함수 구현 ===================== */
/* CO2 핀 연결된 GPIO port 설정 */
/* Timer/Counter2의 관련 Register 설정*/
void init_PWM(){
```

```
        /*- port 초기화
        PWM_OC_DDR  = 0x80;       /*- PB7 Output  */
        PWM_OC_PORT = 0x80;       /*- High 출력  */

        /*- Timer/Counter2 : Fast PWM, 상향 카운터에 CO2를 0으로 클리어, 64k Prescaler  */
        TCCR2 |= 0x6B;

        TCNT2 = 0x00;  /*- 카운터 초기값*/
        OCR2 = 0x00;   /*- TCNT2와 비교되는 값 */
        TIMSK = 0xC0;  /*- Compare Math, Overflow Interrupt 마스킹*/
        TIFR = 0xC0;   /*- Compare Math, Overflow Interrupt 플래그 설정*/

        sei();
}
/* ========================================================== */

/* ==================== Step Motor 초기화 함수 구현 ================== */
/* Step Motor On, 회전 방향 Dir*/
void init_StepMotor(){

        STEP_MOTOR_DDR_01  = 0x80;          /*- Output  PB7(CLK) */
        STEP_MOTOR_PORT_01 = 0x80;

        STEP_MOTOR_DDR_02  = 0x14;          /*- Output   PC4 (DIR), PC2 (ON/OFF)*/
        STEP_MOTOR_PORT_02 = 0x04;
}
/* ========================================================== */

/* ==================== Interrupt Handler 함수 구현 ================== */
/* SIG_OUTOUT_COMPARE2 인터럽트가 발생했을 경우 Interrupt Handlling Vector에 의해 호출 됨*/
SIGNAL(SIG_OUTPUT_COMPARE2){

        PWM_OC_PORT = 0x00;     /* TCNT2 = OCR2 을 경우 OC2에 0(LOW) 출력 */
}
/* ========================================================== */

/* ==================== Interrupt Handler 함수 구현 ================== */
/* SIG_OVERFLOW2 인터럽트가 발생했을 경우 Interrupt Handlling Vector에 의해 호출 됨 */
SIGNAL(SIG_OVERFLOW2){
```

```
        PWM_OC_PORT = 0x80;    /* OC2에 1(High) 출력 */
        TCNT2 = 0x00;          /* TCNT2 에 0xFF -> 0x00 설정 */
        OCR2 = gDutyValue;     /* OCR2에 값 설정 */
}
/* ========================================================== */
```

실습 2. PWM 이용한 Step 모터 속도, 회전 방향 제어하기

▣ PWM_Step02.c (Project Name - PWM_Step02)

```
#include <avr/io.h>
#include <avr/delay.h>
#include <avr/interrupt.h>                /* Interrupt 관련 Header file */

/* ================= PWM & Step 제어를 위한 정의 ================ */
#define PWM_OC_DDR        DDRB      /* PB7 : OC2 Clcok Output Signal */
#define PWM_OC_PORT       PORTB

#define STEP_MOTOR_DDR_01   DDRB    /* PB4(M2), PB5(M1), PB6(M0), PB7(CLK)  */
#define STEP_MOTOR_PORT_01  PORTB

#define STEP_MOTOR_DDR_02   DDRC    /* PC4 (DIR), PC2 (ON/OFF)  */
#define STEP_MOTOR_PORT_02  PORTC
/* ========================================================== */

/* ===================== PWM 제어용 전역변수 ===================== */
unsigned int gDutyValue = 50;             /* 듀티비를 설정하는 전역변수 선언  */
/* ========================================================== */

/* ========================= 함수 선언 ========================= */
void init_PWM();                          /* PWM 관련 port 초기화  */
void init_StepMotor();                    /* Step Motor 관련 port 초기화  */
void pwm_duty(unsigned char nValue);      /* Duty(듀티) 변경 Function */
void change_Dir(unsigned char num);       /* 회전방향 변경 Function */
void change_Speed();                      /* 속도변경 Function */
void pwm_SetpMotor();                     /* 속도/방향 변경 Function */
/* ========================================================== */

/* ==================== PWM 초기화 함수 구현 ==================== */
/* CO2 핀 연결된 GPIO port 설정, Timer/Counter2의 관련 Register 설정 */
void init_PWM()
```

```
{
        /*- port 초기화 */
        PWM_OC_DDR  = 0x80;       /* PB7 Output  */
        PWM_OC_PORT = 0x80;       /* High 출력  */

        /*- Timer/Counter2 : Fast PWM, 상향 카운터에 CO2를 0으로 클리어, 64k Prescaler  */
        TCCR2 |= 0x6B;

        TCNT2 = 0x00;     /* 카운터 초기값 */
        OCR2 = 0x00;      /* TCNT2와 비교되는 값  */
        TIMSK = 0xC0;     /* Compare Math, Overflow Interrupt 마스킹 */
        TIFR = 0xC0;      /* Compare Math, Overflow Interrupt 플래그 설정 */

        sei();
}
/* ============================================================ */

/* ==================== Step Motor 초기화 함수 구현 ==================== */
/*  Step Motor On, 회전 방향 Dir */
void init_StepMotor()
{
        STEP_MOTOR_DDR_01  = 0x80; /* Output  PB7(CLK)  */
        STEP_MOTOR_PORT_01 = 0x80;

        STEP_MOTOR_DDR_02  = 0x14;  /* Output   PC4 (DIR), PC2 (ON/OFF) */
        STEP_MOTOR_PORT_02 = 0x04; /* On, 시계방향 회전  */
}
/* ============================================================ */

/* ======================= Duty(듀티) 변경 함수 구현 ==================== */
/* Duty(듀티) 변경 - OCR2 값 변경 */
void pwm_duty(unsigned char nValue)
{
        OCR2 = nValue;
}
/* ============================================================ */

/* ======================= 회전방향 변경 함수 구현 ======================= */
/* 방향 설정 비트 -> PC4 (DIR) */
void change_Dir(unsigned char num)
```

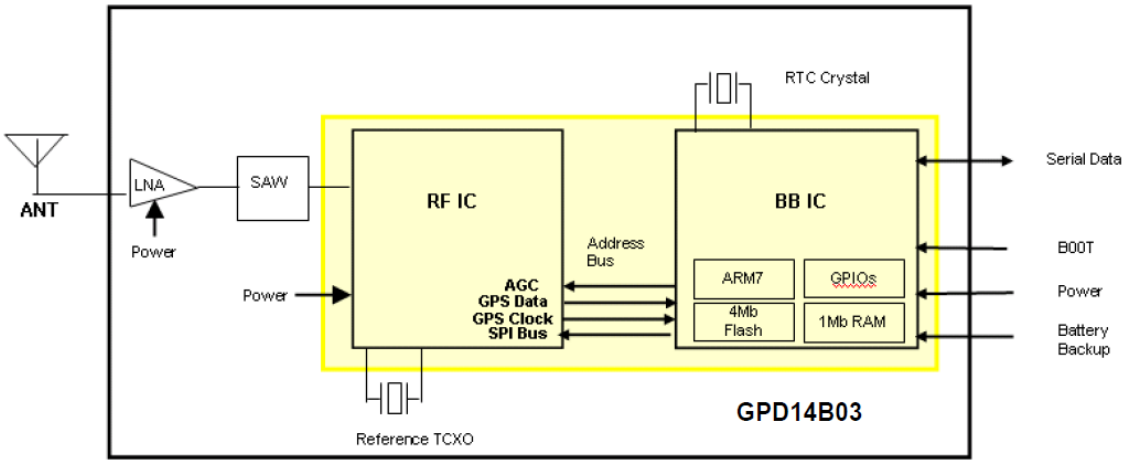
```
{
        if(num)
        {
                STEP_MOTOR_PORT_02  |= 0x10;     /*- 시계 반대 방향 */
        }
        else
        {
                STEP_MOTOR_PORT_02  &= ~0x10;    /*- 시계 방향  */
        }
}
/* ================================================================ */

/* ========================= 속도변경 함수 구현 ======================== */
/* 속도 설정 비트 -> OCR2 에 저장될 값 재설정 */
void change_Speed()
{
        if(gDutyValue >= 0xF0)
          gDutyValue = 5;
        else
          gDutyValue += 50;
}
/* ================================================================ */

/* ======================= 속도/방향 변경 함수 구현 ======================= */
void pwm_SetpMotor()
{
        while(1)
        {
                for(int idx=0; idx<5; idx++)
                {
                        change_Dir(1);      /* 반시계 방향 */
                        _delay_ms(1000);

                        change_Dir(0);      /* 시계 방향 */
                        _delay_ms(1000);
                }
                change_Speed();
        }
}
/* ================================================================ */
```

```
/* ==================== Interrupt Handler Function ==================== */
/* SIG_OUTOUT_COMPARE2 인터럽트 발생시  Interrupt Handlling Vector에 의해 호출 */
SIGNAL(SIG_OUTPUT_COMPARE2)
{
        PWM_OC_PORT = 0x00;    /* TCNT2 = OCR2 을 경우 OC2에 0(LOW) 출력 */
}

/* SIG_OVERFLOW2 인터럽트 발생시 Interrupt Handlling Vector에 의해 호출 */
SIGNAL(SIG_OVERFLOW2)
{
        PWM_OC_PORT = 0x80;    /* OC2에 1(High) 출력 */
        TCNT2 = 0x00;          /* TCNT2 에 0xFF -> 0x00 설정 */
        OCR2 = gDutyValue;     /* OCR2에 값 설정 */
}
/* ============================================================== */

/* ====================== Main 함수 구현 ========================== */
int main()
{
        init_PWM();            /* PWM 관련 port 초기화 */
        init_StepMotor();      /* Step Motor 관련 port 초기화 */
        pwm_SetpMotor();       /* 속도/방향 변경 Function */
        return 0;
}
/* ============================================================== */
```

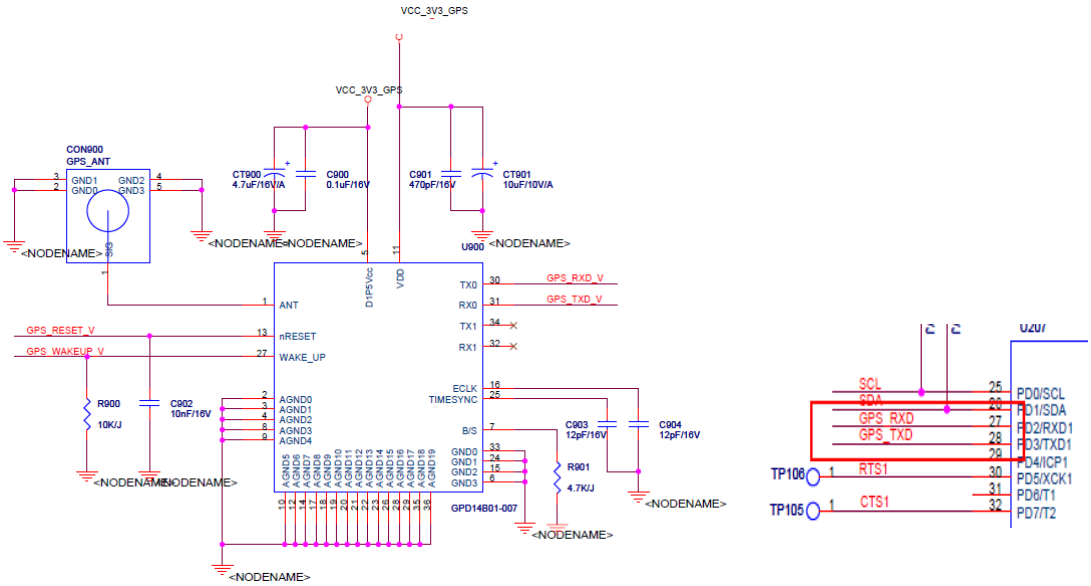
실습 8. GPS 센서 제어하기

8.1 GPS 센서 이해하기

GPS 센서 모듈은 다음과 같이 구성되어 있다.



위의 그림에서 볼 수 있듯이, GPS센서(GPD14B03, Samsung)는 내부에 RF IC와 BB IC를 가지고 있으므로 직접 모듈 내부에서 Data의 처리가 가능하여 실습 보드와 연결된 USART1 Rx 핀으로 데이터를 읽어오기만 하면 된다. 다음은 센서 보드의 간단한 회로도와 실습 보드와 연결된 모습을 나타낸 것이다.



GPS 의 Rx 핀과 Tx 핀은 ATmega128 칩의 Rx1, Tx1 핀에 연결되어 USART1 통신을 이용해 데이터를 주고받을 수 있다. 단, 주의해야 할 점은 GPS 센서 모듈이 통신 속도 9600bps를 사용하므로 USART1 설정 시 유의해야 한다. 자세한 내용은 GPD14B03 모듈의 데이터 시트를 참조하도록 한다.

8.2 GPS 센서 제어 실습하기

실습 1. GPS 센서에서 나오는 데이터 출력하기

GPS 센서에서 나오는 데이터를 하이퍼터미널 화면에 출력하는 소스를 작성하라.

▣ GPS_Ctrl.c (Project name - GPS_Ctrl)

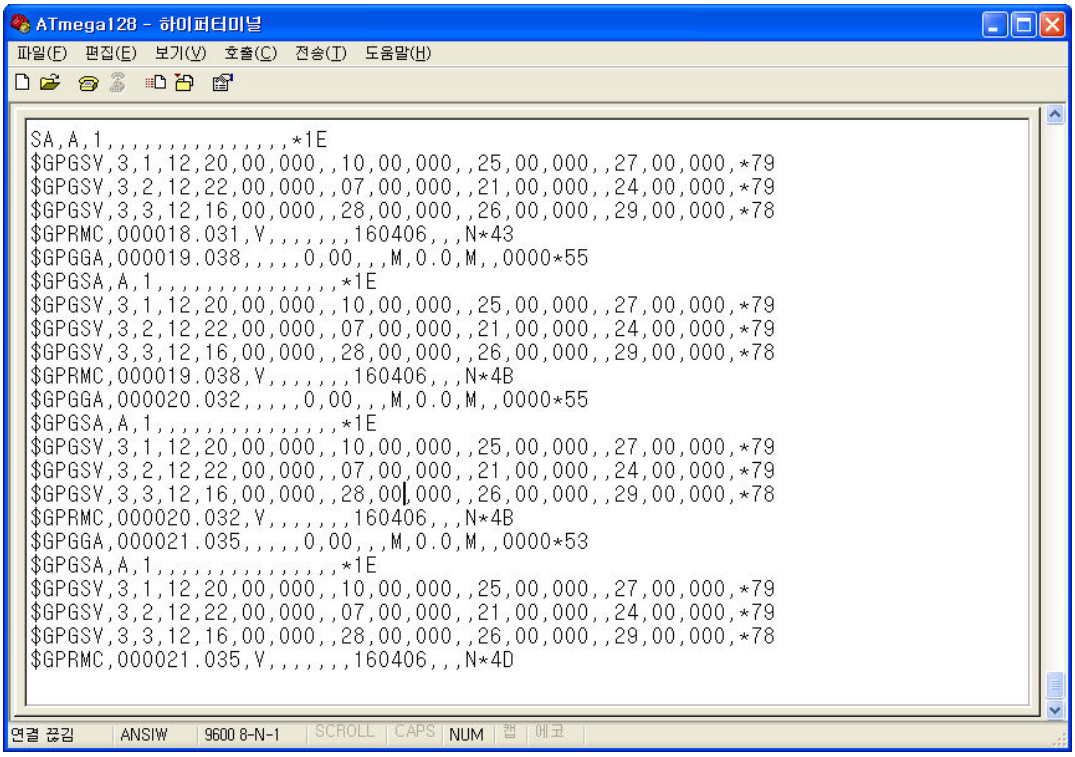
```
#include <avr/io.h>
#include <avr/delay.h>
#include <stdio.h>

/* ==================== USART 제어를 위한 정의 ================== */
#define BAUD_RATE          103                /* 9600 bps */
/* ========================================================== */

/* =============== USART 표준 출력 제어를 위한 정의 ================ */
int GPS_Usart0PutChar(char txData, FILE* stream);
FILE device = FDEV_SETUP_STREAM(GPS_Usart0PutChar, NULL, _FDEV_SETUP_WRITE);
/* ========================================================== */

/* ================ GPS Data를 담을 전역 변수 선언 ================ */
volatile unsigned char rxData;
/* ========================================================== */

/* ================== USART 초기화 함수 구현 ================== */
/* Data 출력을 위한 USART 채널 0 설정 */
void GPS_InitUSART0()
{
        UCSR0B = (1<<TXEN);
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);

        UBRR0L = (unsigned char) BAUD_RATE;
        UBRR0H = (unsigned char) (BAUD_RATE>>8);
}

/* GPS 센서와 통신을 위한 USART 채널 1 설정 */
void GPS_InitUSART1()
{
        UCSR1B = (1<<RXEN) | (1<<TXEN);
        UCSR1C = (1<<UCSZ1) | (1<<UCSZ0);

        UBRR1L = (unsigned char) BAUD_RATE;
```

```
        UBRR1H = (unsigned char) (BAUD_RATE>>8);
}
/* ========================================================== */

/* ===================== GPS 수신 함수 구현====================== */
/* USART1 Receiver 함수 구현 */
void GPS_Usart1Receiver()
{
        while(!(UCSR1A & (1<<RXC))) ;
        rxData = UDR1;
}
/* ========================================================== */

/* ==================== USART 송신 함수 구현 ================== */
/* 표준 출력 함수로 사용하기 위한 함수 구현 */
int GPS_Usart0PutChar(char txData, FILE* stream)
{
        if(txData == '\n')
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\r';
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\n';
        }
        else
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = txData;
        }
}
/* ========================================================== */

/* ===================== Main 함수 구현 ======================== */
/* Receiver 함수를 통해 Data 수신 시 화면 출력 */
int main()
{
        GPS_InitUSART0();
        GPS_InitUSART1();

        stdout = &device;
```

```
        while(1)
        {
                GPS_Usart1Receiver();
                printf("%c", rxData);
        }
}
/* ========================================================== */
```

*** 출력화면**

GPS 센서에서 나오는 데이터를 출력하면 다음과 같은 메시지가 출력된다. 각 메시지에 대한 자세한 내용은 GPD14B03 모듈의 데이터 시트를 참조하도록 한다.



실습 9. 초음파(Distance) 센서 제어하기

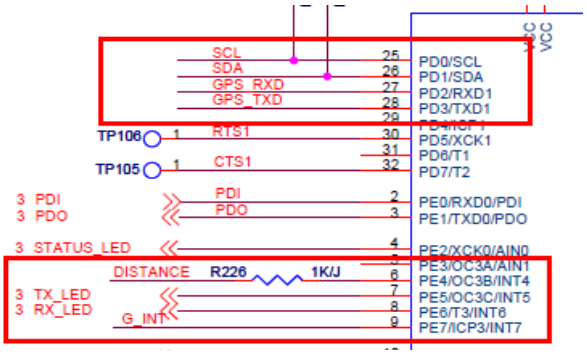
9.1 외부 인터럽트(External Interrupt) 이해하기

외부 인터럽트는 INT7~0 핀에 의해 트리거(trigger) 된다. 만약 인터럽트가 설정되어 있다면 INT7:0 핀이 출력으로 설정되어 있어도 인터럽트가 발생된다.

외부 인터럽트는 외부 인터럽트 제어 레지스터의 설정에 따라 하강 에지(Falling Edge), 상승 에지(Rising Edge), 또는 Low Level 신호에서 발생한다. 단, Low Level 인터럽트로 설정되어 있을 경우 PIN의 값이 'Low' 일 경우 계속해서 인터럽트가 발생하게 되므로 주의한다.

INT7:4 인터럽트의 하강, 상승 에지 인터럽트는 I/O Clock에 의해서 동작할 수 있다. INT3:0 인터럽트의 하강, 상승 에지 인터럽트와 Low Level 인터럽트는 I/O Clock을 사용하지 않는 비동기 검출로, Idle 모드를 제외한 모든 Sleep 모드를 깨우는데 사용된다.

다음 그림은 ATmega128에서 INT7:4 PIN과 INT3:0 PIN을 나타낸 것이다. (INT3:0 은 PD3:0 PIN을 사용한다.)



WinAVR의 라이브러리에서 외부 인터럽트 벡터 값은 다음과 같이 정의되어 있다.

```
/* External Interrupt Request 4 */
#define INT4_vect                    _VECTOR(5)
#define SIG_INTERRUPT4               _VECTOR(5)

/* External Interrupt Request 5 */
#define INT5_vect                    _VECTOR(6)
#define SIG_INTERRUPT5               _VECTOR(6)

/* External Interrupt Request 6 */
#define INT6_vect                    _VECTOR(7)
#define SIG_INTERRUPT6               _VECTOR(7)

/* External Interrupt Request 7 */
#define INT7_vect                    _VECTOR(8)
#define SIG_INTERRUPT7               _VECTOR(8)
```

9.2 외부 인터럽트 레지스터

다음은 외부 인터럽트 제어를 위해 사용되는 레지스터에 대한 설명이다.

1. EICRA, EICRB (External Interrupt Control Register A, B)

EICRA는 INT3:0, EICRB는 INT7:4 인터럽트를 제어하며, INT3:0은 비동기적, INT7:4는 동기 인터럽트이다.

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	
Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
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	

- **ISCn1:0** (External Interrupt n(7:0) Sense Control Bits) : n(7:0)번 인터럽트 PIN에서 어떤 인터럽트를 사용할지 설정한다. n번 인터럽트가 EIMSK 레지스터에서 Enable 되어 있어야 유효하다. 설정 값은 다음과 같다.

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Reserved
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

2. EIMSK (External Interrupt Mask Register)

INT7:0을 개별적으로 허용하는데 사용한다. '1' 로 설정하면 인터럽트 허용, '0' 으로 설정할 경우 인터럽트가 금지된다. 단, SREG 레지스터의 글로벌 인터럽트 허용비트인 I 비트가 '1' 로 설정되어야만 실제로 인터럽트 허용 상태가 되며, 이는 WinAVR 라이브러리에 sei() 함수로 정의되어 있다.

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
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	

- **INTn** : '1' 일 경우 인터럽트 허용, '0' 일 경우 인터럽트 금지

3. EIFR (External Interrupt Flag Register)

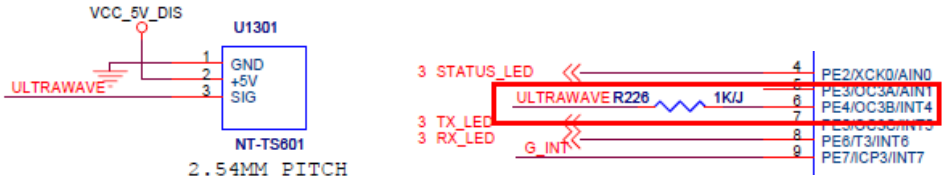
INT7:0 핀에 인터럽트 신호가 입력되어 해당 인터럽트가 트리거(trigger) 되었음을 표시한다. MCU가 인터럽트 벡터를 인출하여 서비스 루틴으로 점프하게 되면 '0' 으로 Clear 된다.

Bit	7	6	5	4	3	2	1	0	
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
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	

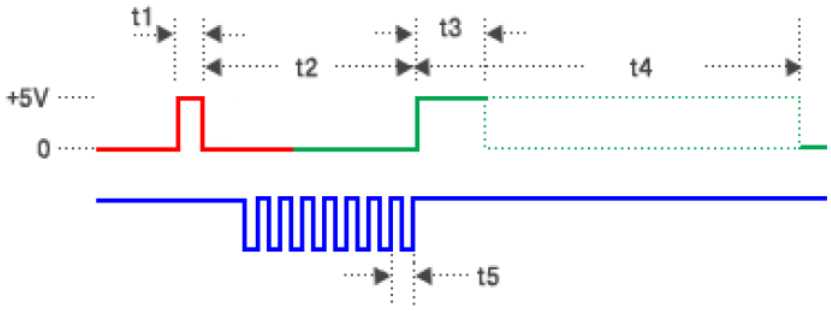
9.3 초음파 센서(NT-TS601) 제어 실습하기

NT-TS601은 비접촉 초음파 측정 기술을 이용한 모듈로, 약 2cm에서 3.3m까지 비교적 정확한 거리를 측정할 수 있다. 또한 하나의 핀을 사용하기 때문에 제어하기가 쉽다.

다음은 NT-TS601의 회로도와 실습 보드에 연결된 Pin을 나타낸 그림이다. 외부 인터럽트 4번 핀에 연결되어 있다.



다음은 NT-TS601 데이터 시트에 나와 있는 동작 타이밍도이다.



	MCU I/O	PARAMETER	SYMBOL	MIN.	TYP.	MAX.	UNIT
	Input	Input trigger pulse	t1	-	5	-	μs
	Output	Echo postpone	t2	500	-	520	μs
		Output echo pulse MIN.	t3	110	-	140	μs
		Output echo pulse MAX.	t4	1.90	-	19.0	ms
	-	Burst pulse cycle	t5	-	25	-	μs

동작 방법은 다음과 같다.

1. MCU에서 출력 핀으로 Input trigger pulse(t1)를 NT-TS601로 보낸다.

2. Input trigger pulse를 받은 NT-TS601은 초음파 출력 부분으로 40kHz의 Burst Pulse(t5)를 발생시키고, MCU는 NT-TS601의 응답을 기다린다(t2).

3. NT-TS601은 Burst Pulse가 물체에 반사되어 되돌아오면 MCU와 연결된 핀으로 Output Echo Pulse를 High로 출력한다(t3).

4. MCU는 Output Echo Pulse를 감지한 후 Output Echo Pulse가 다시 Low로 떨어질 때까지의 시간(t4)을 체크하여 거리를 측정한다.

초음파 센서가 제대로 동작하기 위해서는 물체와의 거리가 최소 2cm, 최대 3.3m 사이여야 하며, 물체가 작거나 초음파와의 각도가 작아 반사되는 초음파를 인식할 수 없는 경우를 피해야 한다.

Output Echo Pulse의 간격에서 얻은 시간 값으로 '속도 = 시간 x 거리' 공식을 이용하여 거리를 구할 수 있다. 단, 음속은 온도의 영향을 받으며, 공식은 다음과 같다.

음파속도 V = 331.5 + 0.60714T [m/s] T : 대기 온도(℃)

따라서 거리를 구하는 공식은 다음과 같이 나타낼 수 있다.

물체와의 거리 (D) [m] = 음속 (V) [m/s] x 시간 (t) [s] = (331.5 + 0.60714T) x (t / 2)

시간 t는 물체에 반사되어 돌아오는 시간을 포함하고 있으므로 실제 Output Echo Pulse의 절반이다. 위의 식을 이용하여 물체와의 거리를 측정할 수 있다.

실습 1. 초음파(Distance) 센서 제어 실습하기

물체와의 거리를 측정하여 하이퍼터미널에 mm 단위로 출력하는 코드를 작성하라.

▣ Distance_Ctrl.c (Project name - Distance_Ctrl)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/delay.h>
#include <stdio.h>

/* ==================== Distance 제어를 위한 정의 ================== */
/* Distance Pin 입출력을 위한 매크로 정의 */
#define   DIST_DDR        DDRE
#define   DIST_PORT       PORTE
#define   DIST_PIN        PINE
#define   DIST_BIT        4
#define   DIST_SET_OUT    (DDRE |= (1<<DIST_BIT))     /* Output */
#define   DIST_SET_IN     (DDRE &= ~(1<<DIST_BIT))    /* Input */
#define   DIST_HIGH       (PORTE |= (1<<DIST_BIT))    /* 1 */
#define   DIST_LOW        (PORTE &= ~(1<<DIST_BIT)) /* 0 */
/* ========================================================== */

/* ================= Timer, USART 제어를 위한 정의 ================ */
#define   CPU_CLOCK              16000000        /* 16MHz */
#define   TICK_PER_SEC           100000          /* 10 micro second */
#define   PRSC_VALUE             8               /* Prescaler 8 */
#define   BAUD_RATE              51              /* Baud rate 19200 bps */
/* ========================================================== */

/* ==================== 음속 제어를 위한 정의 ===================== */
/* 음속을 구하기 위한 온도(실온) 정의 */
#define   TEMP                   25
/* ========================================================== */

/* ================== 음속 제어를 전역 변수 선언 ================== */
/* Timer Counter 변수 선언 */
volatile unsigned int tickCnt;
volatile unsigned int tickEnd;
/* ========================================================== */

/* =============== USART 표준 출력을 위한 변수 선언 ================ */
static int USART0_putchar(char txData, FILE *stream);
```

```
static FILE device=FDEV_SETUP_STREAM(USART0_putchar, NULL, _FDEV_SETUP_WRITE);
/* ========================================================== */

/* =============== Timer/Counter 초기화 함수 구현 ================== */
/* Timer/Counter0 사용을 위한 레지스터 초기화 */
void Dist_InitTimer0Cmp()
{
        TCCR0 = (1<<WGM1) | (1<<CS1);       /* CTC mode, Prescaler 8 */
        TIMSK = (1<<OCIE0);                 /* Compare Match Interrupt Enable */
        TCNT0 = 0;                          /* Counter Init */
        OCR0 = (CPU_CLOCK/TICK_PER_SEC/PRSC_VALUE);       /* 10 micro sec Check */
}
/* ========================================================== */

/* ================== USART 초기화 함수 구현 ==================== */
/* USART0 통신사용을 위한 레지스터 초기화 */
void Dist_InitUSART0()
{
        UBRR0L = (unsigned char) BAUD_RATE;
        UBRR0H = (unsigned char) (BAUD_RATE>>8);
        UCSR0B = (1<<TXEN);                 /* Transmitter Enable */
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);   /* No parity, 1 Stop bit, 8bit Data */
}
/* ========================================================== */

/* ================== Interrupt 처리 함수 구현 ================== */
/* External Interrupt4 사용을 위한 레지스터 초기화 */
void Dist_InitExtInterrupt4()
{
        EICRB |= (1<<ISC41) | (1<<ISC40);   /* INT4 Rising Edge Interrupt */
        EIMSK |= (1<<INT4);                 /* External Interrupt4 Enable */
}
/* ========================================================== */

/* ================== USART 표준 출력 함수 구현 ================== */
/* 표준 출력으로 사용할 USART0_putchar 함수 구현 */
static int USART0_putchar(char txData, FILE *steam)
{
        /* New Line 출력의 경우 '\n\r' 로 바꿔줘야 함 */
        /* (\n : new line, \r : carriage return) */
```

```
        if(txData == '\n')
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\n';
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\r';
        }
        else
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = txData;
        }
        return 0;
}
/* ========================================================== */

/* ================== Interrupt 처리 함수 구현 ==================== */
/* External Interrupt4 Vector 함수 구현 */
SIGNAL(SIG_INTERRUPT4)
{
        /* Rising Edge Interrupt mode일 경우 */
        if(EICRB & ((1<<ISC41) | (1<<ISC40)))
        {
            tickCnt=0;                          /* Start Tick Count */
            EICRB &= ~((1<<ISC41) | (1<<ISC40));  /* Change Low Level Interrupt mode */
        }
        /* Low Level Interrupt mode 일 경우 */
        else
        {
           tickEnd = tickCnt;                 /* End Tick Count */
           EICRB |= (1<<ISC41) | (1<<ISC40);  /* Change Rising Edge Interrupt mode*/
        }
}

/* Timer/Counter0 Compare Match Interrupt Vector 함수 구현 */
SIGNAL(SIG_OUTPUT_COMPARE0)
{
        tickCnt++;
}
/* ========================================================== */
```

```
/* ======================= Main 함수 구현 ======================== */
int main()
{
        unsigned int distVal;

        /* 초기화 함수 호출 */
        Dist_InitTimer0Cmp();
        Dist_InitUSART0();
        Dist_InitExtInterrupt4();

        stdout = &device;           /* 표준출력으로 사용할 장치 설정 */

        while(1)
        {
                cli();                      /* Global Interrupt Disable */
                DIST_SET_OUT;               /* Distance Port Set Output mode */

                DIST_LOW;                   /* Output ‘0’ */
                _delay_us(10);

                DIST_HIGH;                  /* Output ‘1’ */
                _delay_us(10);              /* Keep High Signal for 10us(Min 5us) */
                DIST_LOW;                   /* Output ‘0’ */

                DIST_SET_IN;                /* Distance Port Set Input mode */
                sei();                      /* Global Interrupt Enable */

                _delay_ms(100);             /* Wait Burst Pulse Time */

                /* Calculation Distance Value (us-> /100000, mm-> *1000) */
                distVal = (unsigned int)((331.5 + 0.6*TEMP)
                                * ((float)tickEnd/100000/2) * 1000);
                /* Print Distance Value */
                printf("Distance : %d\n", distVal);
                _delay_ms(1000);
        }
        return 0;
}
/* ========================================================== */
```

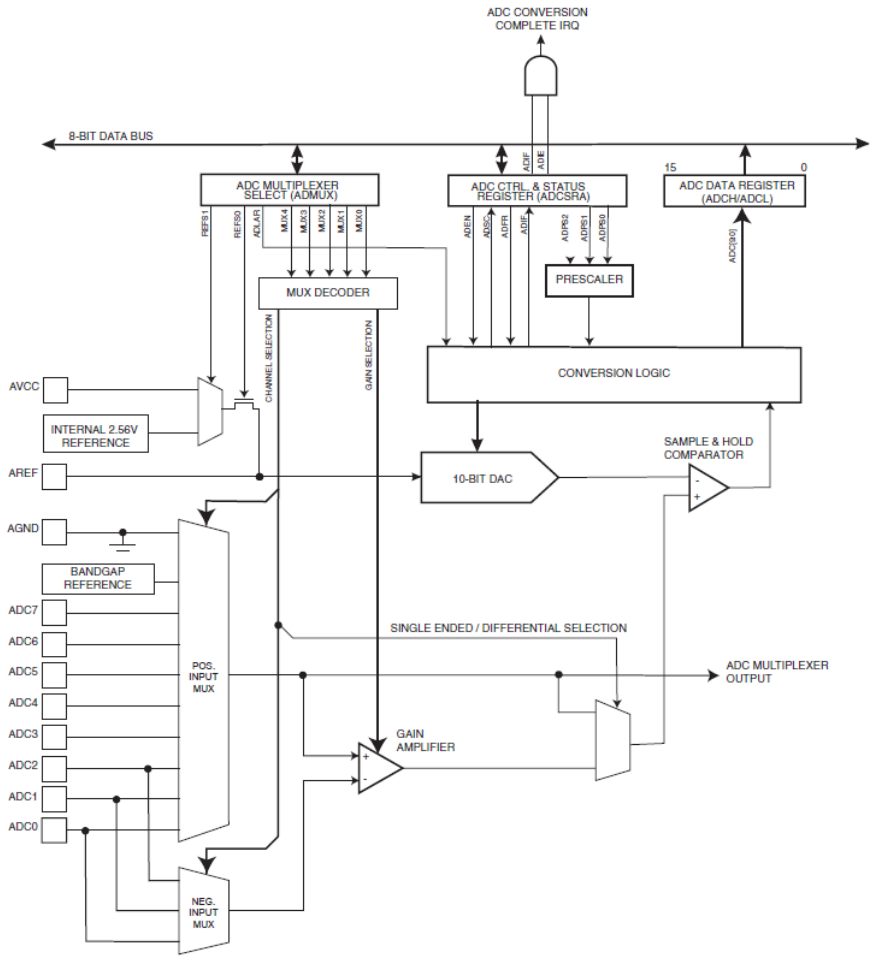
실습 10. 조도(CDS)센서, 연기(Smoke)센서 제어하기

10.1 ADC(Analog Digital Converter) 이해하기

ATmega128에서 제공하는 ADC 기능의 특징은 다음과 같다.

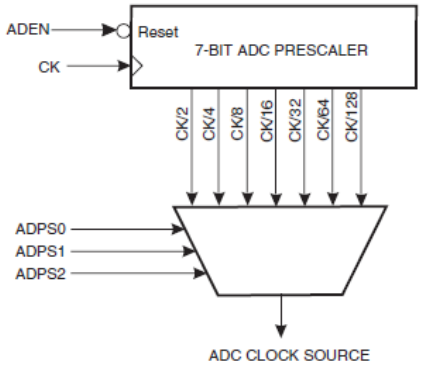
- 10bit 변환 결과
- 13~260㎲ 변환 시간
- 단일 입력 8채널
- 차동 입력 7채널 및 이득 조정이 가능한 2채널
- 결과 값을 왼쪽부터 정렬하도록 설정 가능
- ADC 기준 전압 선택 가능
- ADC 변환 완료 시 인터럽트 발생

다음 그림은 ADC 내부 구조를 블록 다이어그램으로 나타낸 것이다.



- 기준 전원 : A/D 변환에 사용할 기준 전압을 ADMUX[REF1:0]로 선택한다.
- 변환할 채널 : A/D 변환할 채널 및 단일 입력, 혹은 차동 입력 및 이득을 ADMUX[MUX4:0]로 선택한다.
- 변환 결과 : A/D 변환 결과 저장 방법을 ADMUX[ADLAR]로 선택하고, 변환 결과는 ADCH, ADCL로 얻는다.
- 변환 방법 : 변환 방법을 ADCSRA[ADFR], 변환 속도는 ADCSRA[ADPS2:0], 인터럽트 관계는 ADCSRA[ADIE, ADIF], 동작 가능은 ADCSRA[ADEN] 등을 이용하여 제어한다.

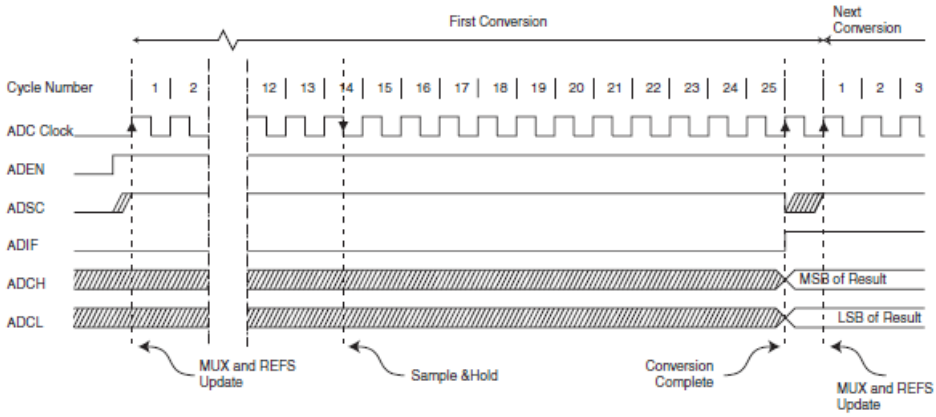
ADC는 다음 그림과 같이 System Clock을 ADC Prescaler 값으로 나눠 사용하며, 보통 50~200kHz의 클록을 사용한다.



Prescaler는 ADEN 비트에 '1' 을 써 넣는 순간부터 동작하게 되며, '0' 이 되면 Reset 된다. ATmega128에서의 ADC 모드는 다음과 같다.

1. 한번 변환 모드 (Single Conversion mode)

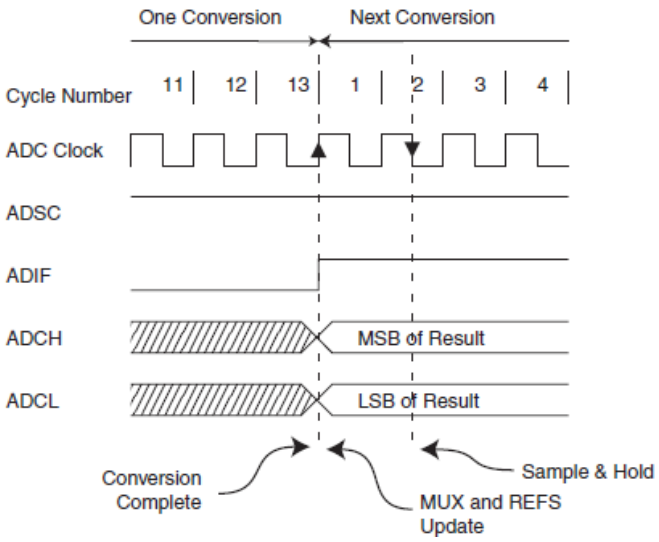
- ADCSRA[ADSC]에 '1' 을 써 넣으면 변환이 초기화 되고, ADC Clock의 다음 상승 모서리에서 처음 변환이 시작된다.
- 다음 타이밍도에서 볼 수 있듯이, 변환 후 첫 번째 변환에서는 Sample&Hold까지 14.5 Cycle의 시간이 소요되며, 다음 변환부터는 1.5 Cycle이 소요된다.
- 변환 시작 후 1.5 ADC Clock Cycle 동안 Sample&Hold가 일어나고, 13 Cycle 후에 변환 결과가 ADCH, ADCL에 저장된다.
- 한번 변환 모드에서 새로운 변환을 시작하기 전에 1개의 Clock이 필요하며, 이 기간 동안 ADCSRA[ADSC]가 '1' 을 유지하고 있으면 즉시 새로운 변환이 시작된다.



2. 계속 변환 모드 (Free Running mode)

ADC 변환 결과가 ADCH, ADCL에 저장된 후 ADCSRA[ADSC]가 '1' 을 유지하고 있으면 즉시 다음 변환이 시작되며, 이를 계속 변환 모드라고 한다.

다음 그림은 계속 변환 모드에서의 ADC 타이밍도이다.



단일 입력 변환일 경우와 차동 입력 변환일 경우에 변환 시간이 달라진다. 다음 표는 첫 번째 변환의 경우와 단일, 차동 입력 변환일 경우 ADC 변환 시간을 Cycle 단위로 나타낸 것이다.

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	14.5	25
Normal conversions, single ended	1.5	13
Normal conversions, differential	1.5/2.5	13/14

- 최초 변환의 경우, 변환이 시작된 후 14.5 Cycle 후에 Sample&Hold가 발생하며, 변환이 완료되는데 총 25 Cycle이 소요된다.
- 단일 입력 변환일 경우, Sample&Hold가 발생할 때까지 1.5 Cycle이 소요되며, 13 Cycle만에 변환이 완료된다.
- 차동 입력 변환일 경우, 1.5/2.5 Cycle만에 Sample&Hold가 발생하며, 13/14 Cycle만에 변환이 완료된다.

10.2 ADC(Analog Digital Converter) 레지스터

다음은 ADC 제어를 위해 사용되는 레지스터에 대한 설명이다.

1. ADMUX (ADC Multiplexer Selection Register)

기준 전압 및 채널을 선택하는데 사용된다.

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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	

- **REFS1:0** (Reference Selection Bits) : 기준 전압을 선택하는데 사용한다.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

- **ADLAR** (ADC Left Adjust Result) : '1' 로 설정될 경우 ADCH, ADCL 레지스터가 왼쪽으로 조정된다.
- **MUX4:0** (Analog Channel and Gain Selection Bits) : 채널 및 이득을 설정하는데 사용한다. 차동 입력일 경우 설정할 수 있는 값이 많지만 여기서는 사용하지 않으므로, 단일 입력을 경우 설정 값만을 살펴보도록 한다.

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			

2. ADCSRA (ADC Control and Status Register A)

ADC 제어 및 ADC의 상태를 나타내는데 사용한다.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

- **ADEN** (ADC Enable) : '1' 을 설정하면 ADC가 동작하며, 동작 중 '0' 을 입력하면 변환이 끝나게 된다.
- **ADSC** (ADC Start Conversion) : 한번 변환 모드일 경우 이 비트에 '1' 을 써 넣으면 변환이 시작되며, 계속 변환 모드일 경우 '1' 을 써 넣으면 첫 번째 변환이 시작된다. 변환 동안 이 비트는 '1' 이 되고, 변환이 끝나면 '0' 이 된다.
- **ADFR** (ADC Free Running Select) : 계속 변환 모드를 설정한다.

- **ADIF** (ADC Interrupt Flag) : ADC 변환이 끝나고 데이터 레지스터가 업데이트될 때 '1' 이 되며, 인터럽트 요청이 허가된 경우 인터럽트를 요청한다.
- **ADIE** (ADC Interrupt Enable) : '1' 일 경우 인터럽트 요청을 허가한다.
- **ADPS2:0** (ADC Prescaler Select Bits) : Prescaler 값을 설정한다. System Clock을 이 값으로 나누어 ADC Clock으로 사용한다.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

3. ADCH, ADCL (ADC Data Register)

변환 결과 값이 저장되며, ADMUX의 ADLAR 비트 값에 따라 저장되는 방법이 다르다. 변환 값이 저장될 때 ADCL에 값이 먼저 저장된 후 ADCH에 값이 저장되므로, ADC의 값을 읽어올 경우 ADCL의 값을 먼저 읽어야 정확한 값을 얻을 수 있다.

- ADLAR이 '0' 일 경우

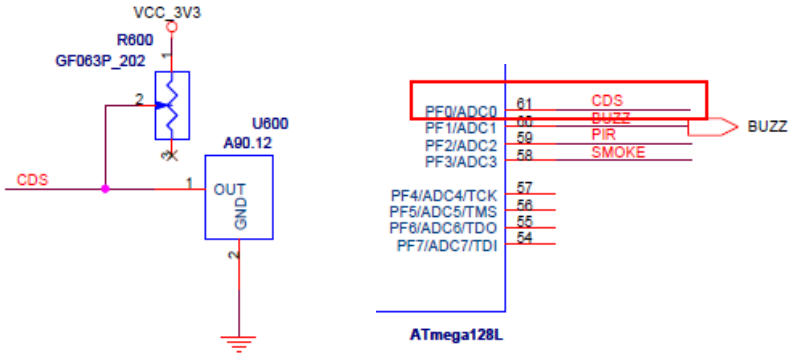
Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- ADLAR이 '1' 일 경우

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

10.3 조도(CDS) 센서 제어 실습하기

CDS 센서는 빛의 양에 따라 달라지는 출력 값을 ADC 변환을 이용하여 측정할 수 있다. 변환 후 10bit 값을 가지므로, 값의 범위는 0~1023이며, 밝기가 밝을수록 낮은 값이 측정된다. 다음 그림은 CDS 센서의 회로도와 실습 보드에 연결된 핀을 나타낸 것이다.



위의 그림에서 볼 수 있듯이, CDS 출력 값은 PF0핀으로 입력되며 ADC0번 채널을 사용하여 ADC 변환 값을 얻을 수 있다.

실습 1. CDS 센서 값 측정하기.

CDS 센서 값을 측정하여 어두울 때 LED가 켜지고, 밝을 때 LED가 꺼지도록 하라.

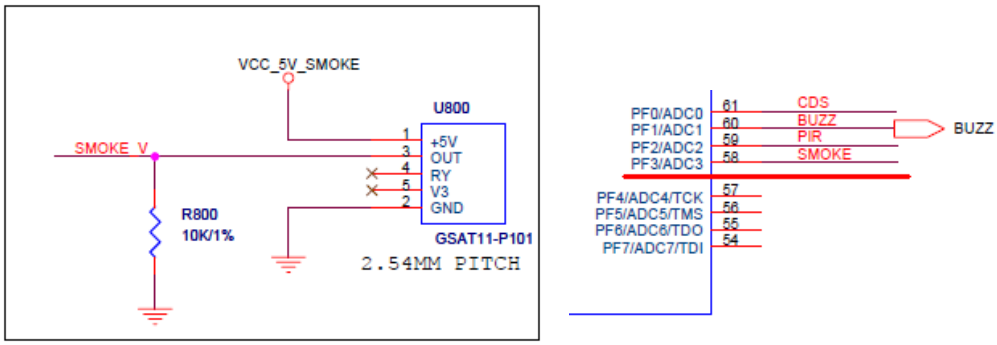
▣ CDS_Ctrl.c (Project name - CDS_Ctrl)

```
#include <avr/io.h>
#include <avr/interrupt.h>

/* ===================== LED 제어를 위한 정의 ==================== */
#define   DDR_LED        DDRE
#define   PORT_LED       PORTE
#define   LED_1          5
#define   CDS_LIMIT      800      /* LED On 되는 한계 값 정의 */
/* ========================================================== */

/* ==================== ADC 초기화 함수 구현 ===================== */
void CDS_InitADC0()
{
        ADMUX = 0;        /* ADC0 채널 설정, ADC 활성화, Prescaler 128 설정  */
        ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (ADPS0);
}
/* ========================================================== */

/* ===================== ADC 변환 함수 구현 ===================== */
unsigned int CDS_StartADConverter()
```

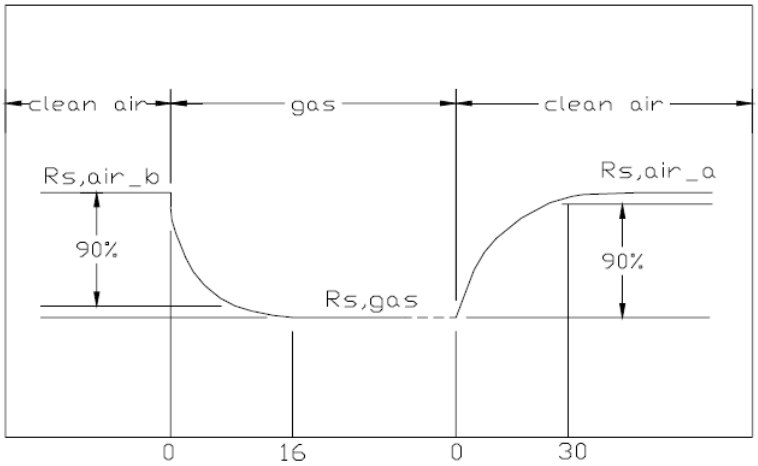
```
{
        unsigned int result;

        cli();                                  /* ADC 변환 중 인터럽트를 금지 */
        ADCSRA |= (1<<ADSC);                    /* ADC 변환 시작 */

        while(!(ADCSRA & (1<<ADIF))) ;          /* ADC 변환 완료 신호를 기다린다. */

        result = ADCL;                          /* 변환 결과 저장 */
        result |= ((ADCH & 0x03) << 8);

        sei();                                  /* 인터럽트 허용 */
        return result;
}
/* ========================================================== */

/* ======================== Main 함수 구현 ======================== */
int main(void)
{
        unsigned int AdcResult;

        CDS_InitADC0();
        DDR_LED |= (1<<LED_1);                  /* LED1을 출력으로 설정 */

        while(1)
        {
                AdcResult = CDS_StartADConverter();  /* ADC 변환 실행 */

                if(AdcResult > CDS_LIMIT)       /* CDS 값이 800보다 클 경우 */
                {
                        PORT_LED |= (1<<LED_1);  /* LED On */
                }
                else                            /* CDS 값이 800보다 작을 경우  */
                {
                        PORT_LED &= ~(1<<LED_1); /* LED Off */
                }
        }
        return 0;
}
/* ========================================================== */
```

10.4 연기(Smoke) 센서(GSAT11-P101) 제어 실습하기

연기(Smoke) 센서는 조도(CDS) 센서와 동작하는 방식이 같으므로, ADC 채널만 변경해주면 된다. 다음 그림은 연기 센서의 회로도와 실습 보드에 연결된 Pin 맵을 나타낸 것이다.



연기 센서의 출력 값은 PF3번 Pin으로 입력되므로 ADC3 채널을 이용하여 값을 측정할 수 있다. 단, GSAT11-P101의 경우 CDS 센서와 달리 값을 측정하기 위해서는 데이터 시트에 명시한 만큼의 Delay가 필요하다. 다음은 GSAT11-P101의 데이터 시트에 나와 있는 반응 시간에 대한 그림이다.



Reaction Time(90%) : Less then 10sec [Between Rs,air_b & Rs,gas]
Recovering Time(90%) : Less then 20sec [between Rs,gas & Rs,air_a]
Beginning stability time : Less then 3 minute

Rs,air_b : Sensor Resistance without gases
Rs,gas : Sensor Resistance after blowing gases
Rs,air_a : Sensor Resistance removing gases

실습 2. 연기(Smoke) 센서 제어 실습하기

연기 센서로 측정한 값을 하이퍼터미널 화면에 출력하라.

▣ Smoke_Ctrl.c (Project name - Smoke_Ctrl)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/delay.h>
#include <stdio.h>

/* ===================== LED 제어를 위한 정의 ==================== */
#define   DDR_LED          DDRE
#define   PORT_LED         PORTE
#define   LED_1            5
/* ============================================================ */

/* ==================== USART 제어를 위한 정의 ================== */
#define   BAUD_RATE        51                        /* 19200 bps */
/* ============================================================ */

/* ================== ADC 제어를 위한 전역 변수 선언 ================ */
/* 변환 결과 값이 저장될 전역 변수 선언 */
volatile unsigned int AdcResult;
/* ============================================================ */

/* ============= USART 표준 출력 제어를 위한 전역 변수 선언 =========== */
static int USART0_putchar(char txData, FILE *stream);
static FILE device=FDEV_SETUP_STREAM(USART0_putchar, NULL, _FDEV_SETUP_WRITE);
/* ============================================================ */

/* ==================== ADC 초기화 함수 구현 ===================== */
void SMOKE_InitADC0()
{
        ADMUX = (1<<MUX1) | (1<<MUX0);              /* ADC3 채널 설정 */
        /* ADC 활성화, Prescaler 128 설정 */
        ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (ADPS0);
}
/* ============================================================ */

/* ==================== USART 초기화 함수 구현 ================== */
void SMOKE_InitUSART0()
{
```

```
        UBRR0L = (unsigned char) BAUD_RATE;
        UBRR0H = (unsigned char) BAUD_RATE >> 8;

        UCSR0B = (1<<TXEN);
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);
}
/* ========================================================== */

/* ==================== USART 표준 출력 함수 구현 ==================== */
/* 하이퍼터미널 화면 출력을 위한 함수 구현 */
/* 표준 출력으로 사용할 USART0_putchar 함수 구현 */
static int USART0_putchar(char txData, FILE *steam)
{
        /* New Line 출력의 경우 '\n\r' 로 바꿔줘야 함 (\n : new line, \r : carriage return) */
        if(txData == '\n')
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\n';
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\r';
        }
        else
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = txData;
        }
        return 0;
}
/* ========================================================== */

/* ==================== ADC 변환 함수 구현 ======================== */
/* ADC0 변환 함수 구현 */
void SMOKE_StartADConverter()
{
        cli();                                  /* ADC 변환 중 인터럽트를 금지 */
        ADCSRA |= (1<<ADSC);                    /* ADC 변환 시작 */

        while(!(ADCSRA & (1<<ADIF))) ;          /* ADC 변환 완료 신호를 기다린다. */

        AdcResult = ADCL;                       /* 변환 결과 저장 */
```

```
        AdcResult |= ((ADCH & 0x03) << 8);

        sei();                                          /* 인터럽트 허용 */
}
/* ========================================================== */

/* ====================== Main 함수 구현 ======================== */
int main(void)
{
        unsigned char i;

        DDR_LED |= (1<<LED_1);                          /* LED1을 출력으로 설정 */

        SMOKE_InitUSART0();
        SMOKE_InitADC0();

        stdout = &device;                       /* 표준출력으로 사용할 장치 설정 */

        /* 3분간의 Beginning Stability Time */
        for(i=0; i<180; i++)
                _delay_ms(1000);

        PORT_LED |= (1<<LED_1);                 /* Complete Stability Time, LED On */

        while(1)
        {
                SMOKE_StartADConverter();
                printf("Smoke Value : %d\n", AdcResult);
                _delay_ms(1000);
        }
        return 0;
}
/* ========================================================== */
```

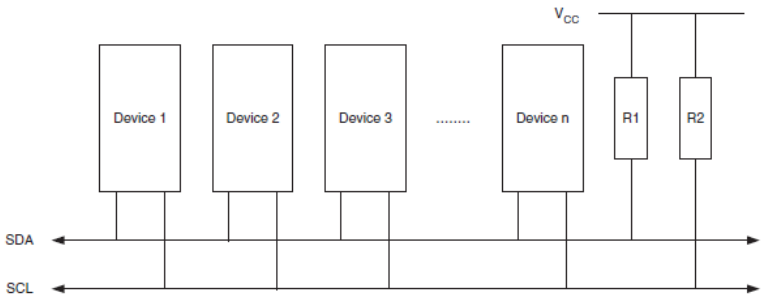
실습 11. G-Sensor 제어하기

11.1 TWI(Two-Wire Serial Interface) 이해

G-Sensor는 SPI 방식과 I2C 방식, 2가지를 지원한다. 실습 보드에서 G-Sensor는 SCL, SDA Pin에 연결되어 있으므로 우리는 I2C 방식으로 G-Sensor를 제어한다.

ATmega128은 직렬 데이터 전송 방법인 I2C(Inter Integrated Circuit)와 같은 TWI 기능을 지원하여, 주변 장치와 동기식으로 데이터 통신을 할 수 있다. TWI(Two-Wire Serial Interface)는 ATmega128과 주변 장치 사이에서 마스터, 혹은 슬레이브 동작을 전이중(Full Duplex) 방법으로 통신하는 방식이다.

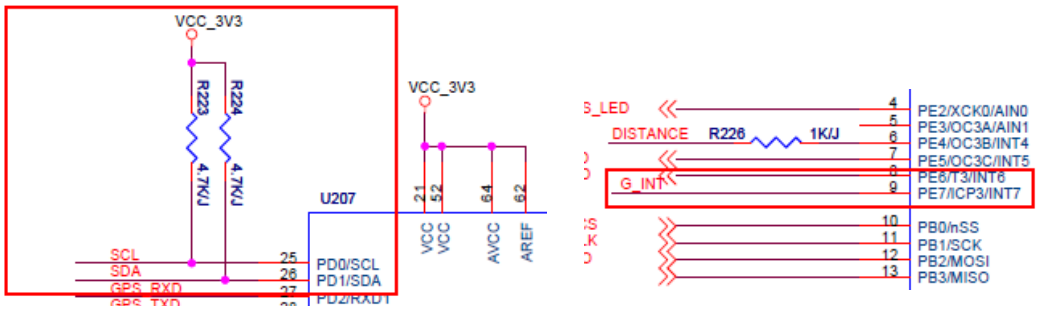
TWI는 하나의 Clock(SCL)과 하나의 데이터(SDA) 선으로 구성된 두 개의 버스 선으로 128개의 서로 다른 장치들을 제어할 수 있다. 아래의 그림은 TWI를 사용하여 연결된 장치들을 나타낸 것이다.



주요 특징은 다음과 같다.

- 두 개의 버스 선만을 사용하는 간단하고 효율적인 통신 방식
- 마스터(Master)와 슬레이브(Slave) 동작을 모두 지원
- 디바이스에서 송수신이 모두 가능
- 7bit 주소를 사용하여 128개의 다른 슬레이브 주소가 가능
- 400kHz 데이터 속도까지 지원

다음 그림은 실습 보드에서 TWI 통신에 이용되는 Pin을 나타내고 있다.



11.2 TWI 레지스터

ATmega128에서는 TWI 통신을 위해 다음과 같은 레지스터를 제공하고 있다.

1. TWBR (TWI Bit Rate Register)

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
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	

TWBR은 TWI의 전송 속도를 설정하는 레지스터이다. 전송 속도는 다음과 같이 계산한다.

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

CMA3000(G-Sensor)의 데이터 시트를 살펴보면 SCL frequency는 400kHz이며, ATmega128의 CPU Clock frequency는 16MHz이므로 TWBR의 값을 계산할 수 있다. TWPS는 이 후에 설명할 TWSR(TWI Status Register)의 TWI Prescaler Bit에 설정되는 값으로, 여기서는 Prescaler를 사용하지 않고 '00'으로 설정할 것이다.

따라서 TWBR의 값은 다음과 같이 계산할 수 있다.

```
TWBR = (CPU_CLOCK / TWI_FREQ - 16) / 2
```

2. TWCR (TWI Bit Rate Register)

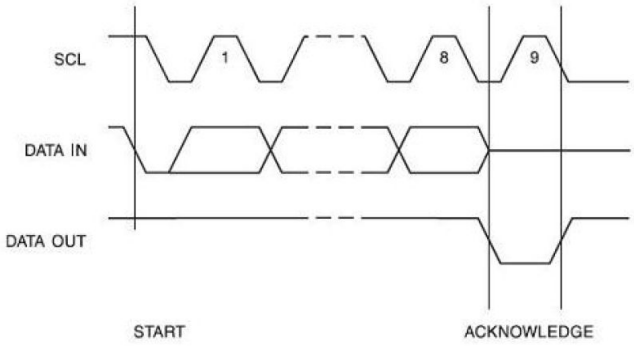
TWI 제어 및 상태를 검사하는데 사용된다.

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **TWINT** (TWI Interrupt Flag) : TWI의 현재 일이 끝나면 하드웨어에 의해 1로 Set 된다. TWCR의 TWIE 비트가 1로 설정되어 있다면 TWI 인터럽트 처리가 완료되면 Clear 된다. 또는 사용자에 의해 '1'이 입력되면 '0'으로 Clear 된다. 사용자는 TWI 작업을 시작하기 위해 이 비트를 Clear 해줘야 한다.
- **TWEA** (TWI Enable Acknowledge Bit) : ACK 신호를 발생시키도록 한다. 이 비트가 '1'로 설정되어 있고 다음과 같은 조건을 만족하면 TWI 버스에 ACK 신호를 전송한다.

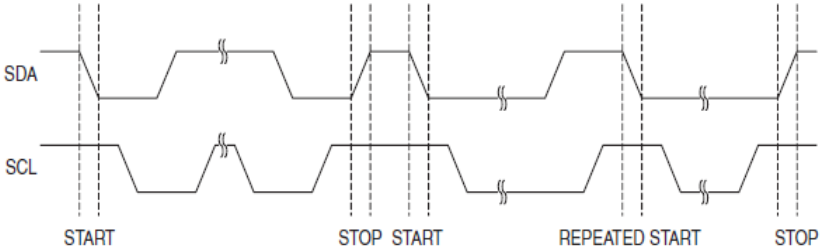
  - 슬레이브 모드일 때, 디바이스가 자신의 주소를 수신했을 경우
  - 마스터 혹은 슬레이브 모드일 때, 수신 모드에서 데이터를 수신했을 경우

일반적으로 ACK 신호는 상대방의 신호에 대한 응답으로 사용되며, 다음 그림은 ACK 신호의 실제 파형이다. SCL이 9번째 High 신호를 줄 때, SDA를 Low로 입력하여 ACK 신호를 보낸다.



- **TWSTA** (TWI Start Condition Bit) : TWI Master로 동작할 때 버스가 비어있다면 Start 조건을 만들어 준다. 비어있지 않다면 Stop 조건이 될 때까지 기다린다. Start 조건이 만들어지면 소프트웨어에 의해 Clear 되어야 한다.

Start 조건과 아래에 나올 Stop 조건의 실제 파형은 아래 그림과 같다. Start 조건은 SCL이 High에 있을 때 SDA가 하강 에지(Falling Edge)를 만드는 것이고, Stop 조건은 SCL이 High에 있을 때 SDA가 상승 에지(Rising Edge)를 만드는 것이다. 시간 간격에 관한 내용은 해당 디바이스의 데이터 시트를 참조하여야 한다.



- **TWSTO** (TWI Stop Condition Bit) : TWSTA와 같이 TWI Master로 동작할 때 Stop 조건을 만들어 준다. 실행되고 나면 하드웨어에 의해 자동으로 Clear 된다.
- **TWEN** (TWI Enable Bit) : TWI 동작을 활성화시키기 위해 '1' 로 설정해야 한다. '0' 으로 설정할 경우 TWI의 모든 동작이 멈추게 된다.
- **TWIE** (TWI Interrupt Enable Bit) : 글로벌 인터럽트가 활성화 되어있고 이 비트가 '1' 이면, TWINT 비트가 '1' 로 될 때 TWI 인터럽트가 발생한다.

3. TWSR (TWI Status Register)

TWI의 상태를 검사하는데 사용하며, TWS7~ TWS3은 읽기만 가능하다.

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- **TWS7:3** (TWI Status) : TWI의 상태를 나타내며, 자세한 내용은 데이터 시트를 참조한다.
- **TWPS1:0** (TWI Prescaler Bits) : TWI 속도 설정을 위해 분주비를 설정하기 위한 비트이다. 이 값을 참조하여 앞에서 설명했던 TWBR의 값이 결정된다. 설정 값은 다음과 같으며, 여기서는 분주하지 않고 '00' 으로 설정하여 사용한다.

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

4. TWDR (TWI Data Register)

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

TWDR에는 송수신용 데이터를 담고 있는 레지스터이다. 송신 모드에서는 전송할 다음 바이트 데이터가 있고, 수신 모드에서는 마지막에 받은 바이트 데이터가 담겨있다.

Master 모드일 경우 Slave의 주소를 전송할 때, 7~1번 bit는 7bit 주소 값을, 0번 bit는 R/W를 설정하는 값으로 사용된다. (G-Sensor에서는 Read : '0', Write : '1')

5. TWAR (TWI (Slave) Address Register)

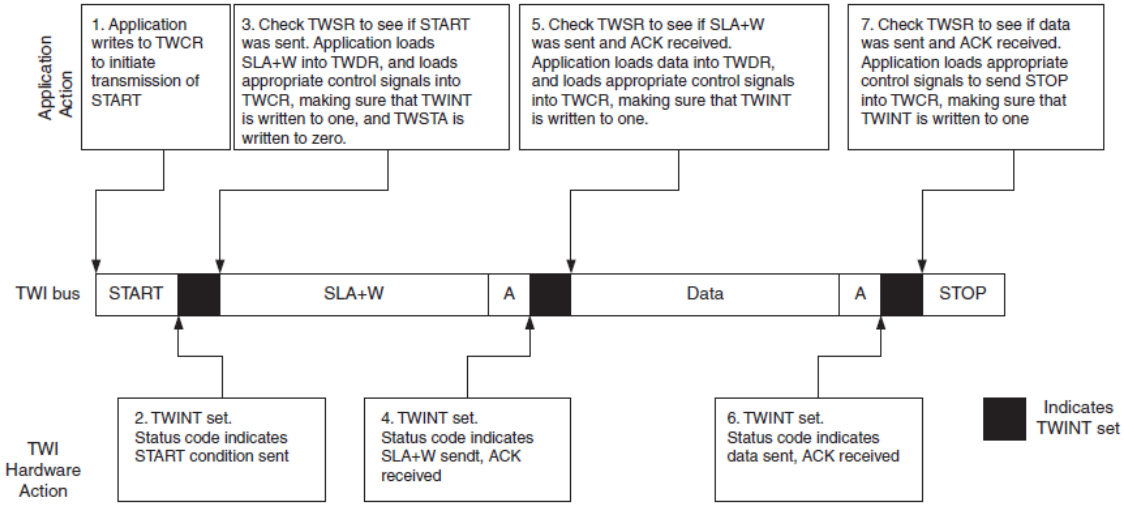
Slave 모드에서 자신의 7bit 주소 값이 저장된다. Master 모드에서는 사용하지 않는다.

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

- **TWA6:0** (TWI (Slave) Address) : 7bit의 슬레이브 주소 값이 들어간다.
- **TWGCE** (TWI General Call Recognition Enable bit) : 범용 콜 응답을 활성화 시킨다. Master에서 디바이스 전체와 통신하고자 할 경우 이 비트가 '1'이 설정되어 있어야 응답할 수 있다.

11.3 TWI 제어하기

다음 그림은 ATmega128 데이터 시트에 나와 있는 TWI 동작 순서이다.



1. Start 조건을 전송한다. TWCR의 Start Bit를 Set한 후, TWINT 비트를 Clear 해야 TWI가 동작한다. 주의해야 할 점은 TWCR의 TWINT 비트에는 Not Gate가 붙어있기 때문에, TWINT에 '1'을 넣어야 '0'으로 Clear 된다는 것이다. 즉, 사용자는 TWCR에 TWSTA를 '1'로 Set하고, TWINT를 '1'로 Clear 함으로써 Start 조건을 전송할 수 있다. (TWEN 비트는 항상 '1'로 설정되어 있어야 한다.)

2. Start 조건 전송이 완료되기를 기다린다. TWINT 비트가 '1'이 되면 전송이 완료된 것을 의미한다.

3. Start 조건 전송이 완료되었으면 TWDR에 Slave의 주소와 R/W 값을 넣어 전송한다. TWCR의 Start Bit가 자동으로 Clear되지 않으므로 소프트웨어적으로 Clear 해줘야 하며, TWINT를 '1'로 Clear하는 것으로 데이터를 전송할 수 있다.

4. ACK 신호를 기다린다. TWINT 비트가 '1'이 되면 ACK 신호를 받은 것을 의미한다.

5. ACK 신호를 받은 것으로 주소 값이 제대로 전송된 것을 확인하면 TWDR에 실제 데이터를 넣어 전송한다. 마찬가지로 TWINT를 '1'로 Clear하여 전송한다.

6. ACK 신호를 기다린다.

7. ACK 신호가 확인되면 Stop Bit를 전송한다. TWCR의 Stop Bit인 TWSTO를 '1'로 Set 한 후, TWINT를 '1'로 Clear 함으로써 Stop 조건을 전송할 수 있다. 단, Stop Bit는 Start Bit와 달리 하드웨어에서 자동으로 Clear 한다.

기본적인 전송 방법은 위에서 설명한 것과 비슷하지만, 각각의 디바이스마다 전송하는 방법이 약간씩 다를 수 있으며 Slave 주소도 물론 다르다. 데이터를 읽어오는 방법 역시 디바이스마다 다를 수 있으므로 자세한 내용은 해당 디바이스의 데이터 시트를 참고해야 한다.

다음은 위에서 설명한 전송 과정을 C 예제 소스와 함께 간단히 표로 만든 것이다. 우리가 사용할 G-Sensor의 경우 아래 과정과 크게 다르지 않으므로 참조하도록 한다.

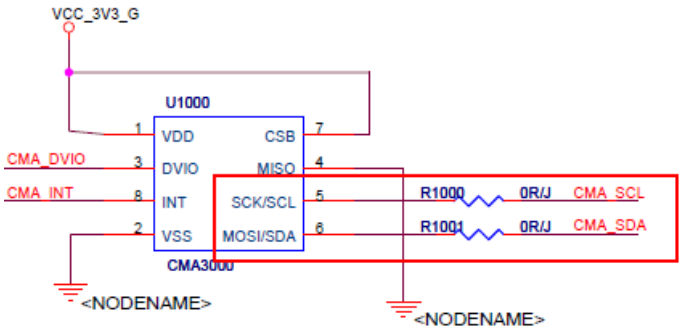
번호	C 예제 소스	설명
1	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTA);	Start 조건을 내보낸다.
2	while(!(TWCR & (1<<TWINT))) ;	TWINT가 '1'이 되기를 기다린다. 여기서는 Start 조건이 완료된 것을 의미한다.
3	TWDR = SLA_W; TWCR = (1<<TWINT) | (1<<TWEN);	TWDR에 Slave의 주소를 써넣는다. TWCR은 TWSTA를 Clear하고 TWINT를 Clear하기 위해 새로 써넣는다.
4	while(!(TWCR & (1<<TWINT))) ;	TWINT가 '1'이 되기를 기다린다. 여기서는 Slave 주소가 전송 완료되어 ACK 신호를 받은 것을 의미한다.
5	TWDR = DATA; TWCR = (1<<TWINT) | (1<<TWEN);	TWDR에 실제로 전송하기 위한 Data를 넣은 후 TWINT를 Clear하여 전송한다.
6	while(!(TWCR & (1<<TWINT))) ;	TWINT가 '1'이 되기를 기다린다. 여기서는 Data가 전송 완료되어 ACK 신호를 받은 것을 의미한다.
7	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);	Stop 조건을 내보낸다.

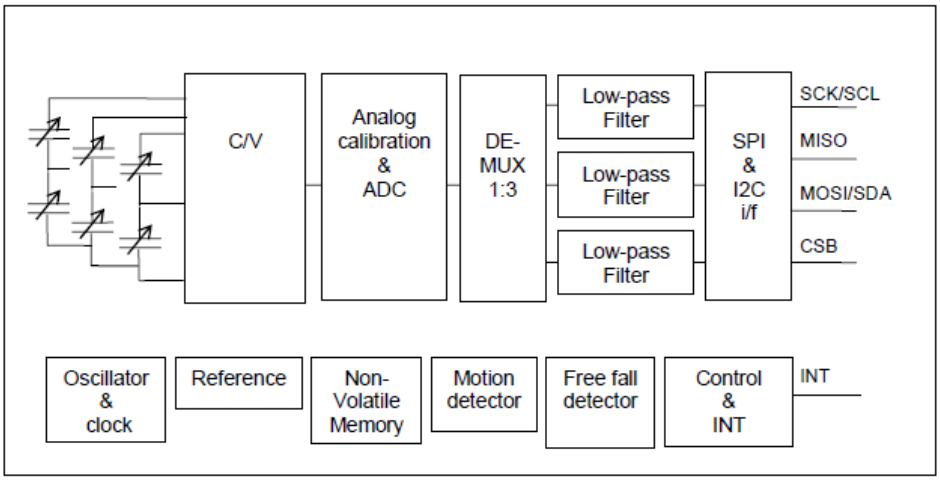
다음은 데이터를 읽어오는 부분만을 설명한 것이다.

ACK를 보내야 할 때	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<(TWEA); while(!(TWCR & (1<<TWINT))) ; DATA = TWDR;	ACK를 보내기 위해 TWCR에 TWEA 비트를 '1'로 Set한다. 데이터가 수신 완료되어 TWINT 비트가 '1'로 Set되면 TWDR에서 Data를 읽어온다.
ACK를 보내지 않을 때	TWCR = (1<<TWINT) | (1<<TWEN); while(!(TWCR & (1<<TWINT))) ; DATA = TWDR;	TWEA를 '1'로 설정하지 않고 데이터를 수신한다.

11.4 G-Sensor(CMA3000) 제어 실습하기

실습에 사용되는 G-Sensor(가속도 센서) 칩은 CMA3000을 사용하며, 다음은 그 회로도와 내부 블록 다이어그램이다.





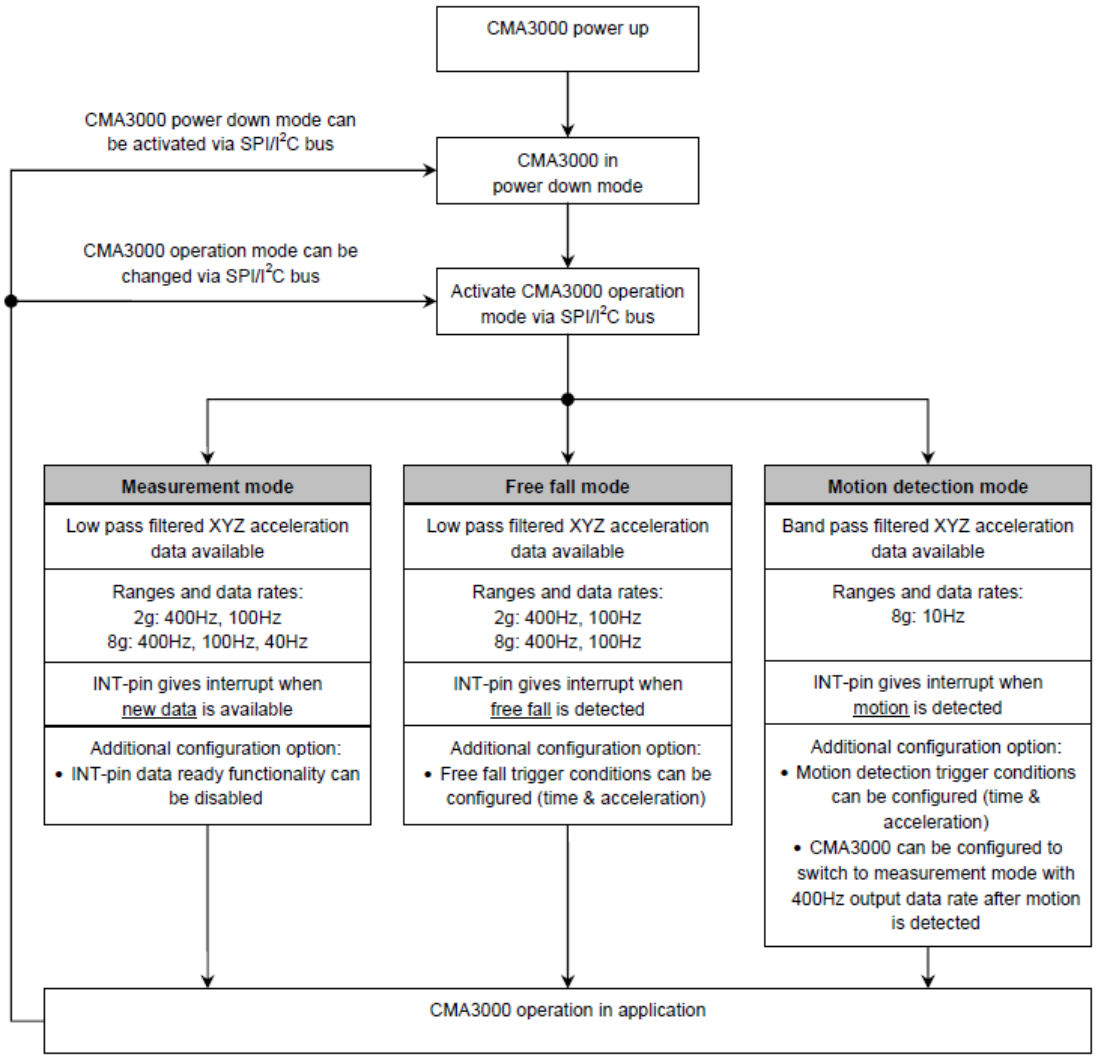
5, 6번 핀의 SCL, SDA 핀은 ATmega128 칩의 SCL, SDA 핀과 연결되어 있다. 따라서 ATmega128에서 제공하는 TWI 통신을 이용하여 데이터를 주고받을 수 있다.

다음 표는 CMA3000의 데이터 시트에 나와 있는 특징들이다.

Features	CMA3000-D01
Supply voltage	1.7 V – 3.6 V
I/O voltage	1.7 V – 3.6 V
Measuring range (selectable)	±2 g, ±8 g
Resolution (2g /8g range)	17 mg / 67 mg
Sensitivity (2g /8g range)	56 counts/g / 14 counts/g
Motion detection	User enabled
Free fall detection	User enabled
Interface	SPI max 500 kHz, I^2C fast mode 400 kHz
Clock	Internal

1. CMA3000 동작 모드

다음 그림은 CMA3000의 동작 Flow Chart이다. CMA3000은 세 가지 모드를 가지며, SPI 또는 I2C(TWI) 통신을 이용하여 데이터를 주고받는다.



- Measurement mode : X, Y, Z 축의 값이 바뀔 때마다 인터럽트를 발생시켜 사용자에게 알린다. 사용자가 측정 범위 및 Sample rate를 설정할 수 있으며, 이 데이터는 Motion detection mode에 활용될 수 있다.
- Free-fall mode : 자유낙하 감지 모드. 자유 낙하가 감지되면 인터럽트를 발생시켜 사용자에게 알린다.
- Motion detection mode : 움직임 감지 모드. 움직임이 감지되면 인터럽트를 발생시킨다. 각 모드에 관한 자세한 설명은 데이터 시트를 참조한다.

2. CMA3000 레지스터

다음은 CMA3000에서 제공하는 레지스터와 주소 값을 나타낸 것이다.

Address	Name	Description	Mode (R, RW, NV)	Reg. type
00h	**WHO_AM_I**	Identification register	**R**	Output
01h	**REVID**	ASIC revision ID, fixed in metal	**R**	Output
02h	**CTRL**	Configuration (por, operation modes)	**RW**	Conf
03h	**STATUS**	Status (por, EEPROM parity)	**R**	Output
04h	**RSTR**	Reset Register	**RW**	Conf
05h	**INT_STATUS**	Interrupt status register	**R**	Output
06h	**DOUTX**	X channel output data register	**R**	Output
07h	**DOUTY**	Y channel output data register	**R**	Output
08h	**DOUTZ**	Z channel output data register	**R**	Output
09h	**MDTHR**	Motion detection threshold value register	**RW**	Conf
0Ah	**MDFFTMR**	Free fall and motion detection time register	**RW**	Conf
0Bh	**FFTHR**	Free fall threshold value register	**RW**	Conf
0Ch	**I2C_ADDR**	I^2C device address	**R**	Conf
0Dh-19h		Reserved		

Address is the register address in hex format.
RW – Read / Write register, R – Read-only register, NV – non-volatile register content.

맨 오른쪽에 나오는 Address 값으로 각 레지스터의 값을 읽어올 수도, 쓸 수도 있다. 자세한 내용은 CMA3000 데이터 시트에서 확인할 수 있으며, 여기에서는 중요한 몇 가지만 살펴보도록 한다.

- **CTRL** (Control Register) : G-Range 및 모드 설정, 인터럽트 활성화 등을 설정할 수 있다.

Address: **02h**
Register name: **CTRL**, Control register

Bits	Mode	Initial Value	Name	Description
7	RW	0	G_RANGE	0 – 8g measurement range is selected 1 – 2g measurement range is selected
6	RW	0	INT_LEVEL	0 – INT is active when INT pin is set to logic high 1 – INT is active when INT pin is set to logic low
5	RW	0	MDET_EXIT	0 – Device goes to measurement mode after motion is detected (400Hz ODR) 1 – Device remains in motion detection mode after motion is detected.
4	RW	0	I2C_DIS	0 – I^2C interface enabled 1 – I^2C interface disabled.
3:1	RW	0	MODE[2:0]	000 – Power down mode, default mode. 001 – Measurement mode, 100 Hz ODR. 010 – Measurement mode, 400 Hz ODR. 011 – Measurement mode, 40 Hz ODR. 100 – Motion detection mode, 10 Hz ODR. 101 – Free fall detection mode, 100 Hz ODR. 110 – Free fall detection mode, 400 Hz ODR. 111 – Power down mode
0	RW	0	INT_DIS	0 – Interrupts enabled • Measurement mode: data ready • Motion detection mode: motion detected • Free fall detection mode: free fall detected 1 – Interrupts disabled

Note that after changing MODE bits it may take some time to recover the target operating state.
ODR = Output Data Rate.

- **DOUTX, DOUTY, DOUTZ** (Data Out X, Y, Z) : X, Y, Z 축의 값이 저장되어 있다.

Address: **06h**
Register name: **DOUTX**, X-channel output register

Bits	Mode	Initial Value	Name	Description
7:0	R	0h	DOUTX	See SPI data frame description for more info.

Address: **07h**
Register name: **DOUTY**, Y-channel output register

Bits	Mode	Initial Value	Name	Description
7:0	R	0h	DOUTY	See SPI data frame description for more info.

Address: **08h**
Register name: **DOUTZ**, Z-channel output register

Bits	Mode	Initial Value	Name	Description
7:0	R	0h	DOUTZ	See SPI data frame description for more info.

3. CMA3000 I2C 통신

I2C는 2-wire serial interface(TWI)이며, CMA3000은 Master-Slave 모드에서 항상 Slave 모드여야 한다. I2C를 사용하기 위해 CMA3000의 디바이스 주소 값은 0011100b로 정의되어 있다. 7bit 주소를 사용하며, 1채널만을 사용한다. Standard mode와 Fast mode를 지원한다.

- I2C Write mode

I2C Write mode는 Master에서 8bit(Address 7bit + R/W) Slave Address를 전송한 후, 8bit Register Address를 전송한 다음 데이터를 전송하여 값을 쓸 수 있다.

- I2C Read mode

Read mode 역시 Master에서 8bit(Address 7bit + R/W) Slave Address를 전송한 후, 8bit Register Address를 전송하면 Slave로부터 8bit의 데이터를 수신 받을 수 있다.

다음은 I2C 통신의 예를 나타낸 것으로, Device Address는 0011100b, DOUTX 값을 읽어오는 예와 MDFFTIMR 레지스터에 값을 설정하는 예이다.

CASE 1: I2C 8 bit read from register DOUTX

S	device addr byte 0011100	0	SA	register address byte 00000110	SA	RS	device addr byte 0011100	1	SA	register data MSB first	MA	E

CASE 2: I2C 8 bit write to register MDFFTIMR

S	device addr byte 0011100	0	SA	register address byte 00001010	SA	register data to write MSB first	SA	E

S = Start condition
RS = Repeated Start Condition
E = End condition
SA = Slave acknowledgement
MA = Master acknowledgement

실습 1. G-Sensor Measurement mode Test

I2C 통신으로 G-Sensor의 x, y, z 값을 읽어 하이퍼터미널 화면에 출력하라.

▣ GSensor_Ctrl.c (Project name - GSensor_Ctrl)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/delay.h>
#include <stdio.h>

/* ====================== Slave 주소  선언 ===================== */
#define   SLV_ADDR        0x1C                    /* 0011100b */
#define   SLV_ADDR_W      (SLV_ADDR << 1)         /* Write : 0 (00111000b) */
#define   SLV_ADDR_R      ((SLV_ADDR << 1) + 1)   /* Read : 1 (00111001b) */
/* ========================================================== */

/* ======================== 레지스터 주소 선언 ==================== */
#define   WHO_AM_I        0x00
#define   REVID           0x01
#define   CTRL            0x02
#define   STATUS          0x03
#define   RSTR            0x04
#define   INT_STATUS      0x05
#define   DOUTX           0x06
#define   DOUTY           0x07
#define   DOUTZ           0x08
#define   MDTHR           0x09
#define   MDFFTMR         0x0A
#define   FFTHR           0x0B
#define   I2C_ADDR        0x0C
/* ========================================================== */

/* ==================== Clock 제어를 위한 선언 ==================== */
#define    CPU_CLOCK      16000000
#define   SCL_FREQUENCY   400000          /* 400kHz */
#define   BAUD_RATE       51              /* Baud Rate for USART Communication */
/* ========================================================== */

/* =============== 데이터를 저장할 전역 변수 선언 ==================== */
/* x, y, z 데이터를 저장할 전역 변수 선언 */
volatile unsigned char dataX;
volatile unsigned char dataY;
```

```
volatile unsigned char dataZ;
volatile unsigned char iFlag;            /* 인터럽트 발생 플래그 변수 선언 */
/* ========================================================== */

/* ================ USART 표준 출력 전역 변수 선언 ================== */
static int USART0_putchar(char txData, FILE *stream);
static FILE device=FDEV_SETUP_STREAM(USART0_putchar, NULL, _FDEV_SETUP_WRITE);
/* ========================================================== */

/* ================= TWI 통신 초기화 함수 구현 ==================== */
/* TWI 통신을 위한 레지스터 초기화 */
void GS_InitTWI()
{
        TWBR = (CPU_CLOCK / SCL_FREQUENCY - 16) / 2;    /* 통신 속도 설정 */
        TWSR &= ~((1<<TWPS1) | (1<<TWPS0));             /* 00, 분주비 사용하지 않음 */
}
/* ========================================================== */

/* ================= USART통신 초기화 함수 구현 ==================== */
/* USART 통신을 위한 레지스터 초기화 */
void GS_InitUSART0()
{
        UBRR0H = (unsigned char) BAUD_RATE >> 8;
        UBRR0L = (unsigned char) BAUD_RATE;

        UCSR0B = (1<<TXEN);                 /* Transmitter 활성화 */
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);   /* No Parity, 1 Stop bit, 8bit data */
}
/* ========================================================== */

/* ==================== Interrupt 처리 함수 구현 ==================== */
/* 외부 인터럽트 사용을 위한 레지스터 초기화 */
/* Enable External Interrupt7 */
void GS_InitExtInterrupt7()
{
        EICRB = ~((1<<ISC71) | (1<<ISC70));    /* Low Level Detected */
        EIMSK = (1<<INT7);
        EIFR = 0x00;
}
/* ========================================================== */
```

```
/* ================== USART 표준출력 함수 구현 ================== */
/* USART Transmitter 구현 */
/* 표준 출력으로 사용할 USART0_putchar 함수 구현 */
static int USART0_putchar(char txData, FILE *steam)
{
	/* New Line 출력의 경우 '\n\r' 로 바꿔줘야 함 (\n : new line, \r : carriage return) */
	if(txData == '\n')
	{
		while(!(UCSR0A & (1<<UDRE))) ;
		UDR0 = '\n';
		while(!(UCSR0A & (1<<UDRE))) ;
		UDR0 = '\r';
	}
	else
	{
		while(!(UCSR0A & (1<<UDRE))) ;
		UDR0 = txData;
	}
	return 0;
}
/* ========================================================== */

/* ================== I2C 통신 Read 함수 구현 ================== */
unsigned char GS_TwiReadData(unsigned char regAddr)
{
	unsigned char rxData;

	/* Start 조건을 만들어 전송한다. */
	TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
	while(!(TWCR & (1<<TWINT))) ;

	/* Slave 주소 값과 Write bit를 전송한다. */
	TWDR = SLV_ADDR_W;
	TWCR = (1<<TWINT) | (1<<TWEN);
	while(!(TWCR & (1<<TWINT))) ;

	/* 레지스터 주소 값을 전송한다. */
	TWDR = regAddr;
	TWCR = (1<<TWINT) | (1<<TWEN);
```

```
        while(!(TWCR & (1<<TWINT))) ;

        /* 다시 Start 조건을 만들어 전송. 값을 읽어오기 위해 Slave의 주소 값을 한 번 더 전송해야 한다. */
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT))) ;

        /* Slave 주소 값을 Read bit와 함께 전송한다. */
        TWDR = SLV_ADDR_R;
        TWCR = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT))) ;

        /* ACK 신호를 만들고, 데이터가 수신되기를 기다린다. */
        TWCR = (1<<TWINT) | (1<<TWEA) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT))) ;

        /* 수신이 완료되면 데이터 레지스터의 값을 읽어온다. */
        rxData = TWDR;

        /* Stop 조건을 만들어 전송한다. */
        TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);

        return rxData;
}
/* ========================================================== */

/* ================== I2C 통신 Write 함수 구현 ================== */
void GS_TwiWriteData(unsigned char regAddr, unsigned char wData)
{
        /* Start 조건을 만들어 전송한다. */
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT))) ;

        /* Slave 주소 값과 Write bit를 전송한다. */
        TWDR = SLV_ADDR_W;
        TWCR = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT))) ;

        /* 읽어 올 레지스터의 주소 값을 전송한다. */
        TWDR = regAddr;
        TWCR = (1<<TWINT) | (1<<TWEN);
```

```
        while(!(TWCR & (1<<TWINT))) ;

        /* 실제 데이터를 전송한다. */
        TWDR = wData;
        TWCR = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT))) ;


        /* Stop 조건을 만들어 전송한다. */
        TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);

        _delay_ms(10);
}
/* ============================================================ */

/* ==================== Interrupt 처리 함수 구현 ===================== */
/* 인터럽트 처리 함수 구현 */
/* X, Y, Z 값을 읽어 와 USART 통신을 이용해 하이퍼터미널에 출력한다. */
SIGNAL(SIG_INTERRUPT7)
{
        cli();

        dataX = GS_TwiReadData(DOUTX);
        dataY = GS_TwiReadData(DOUTY);
        dataZ = GS_TwiReadData(DOUTZ);

        iFlag = 1;          /* 인터럽트 발생 Flag Set */
        sei();
}
/* ============================================================ */

/* ======================= Main 함수 구현 ======================== */
/* main 함수 구현 */
/* 레지스터 초기화, 모드 설정, 글로벌 인터럽트 활성화 */
int main(void)
{
        GS_InitTWI();
        GS_InitUSART0();
        GS_InitExtInterrupt7();
```

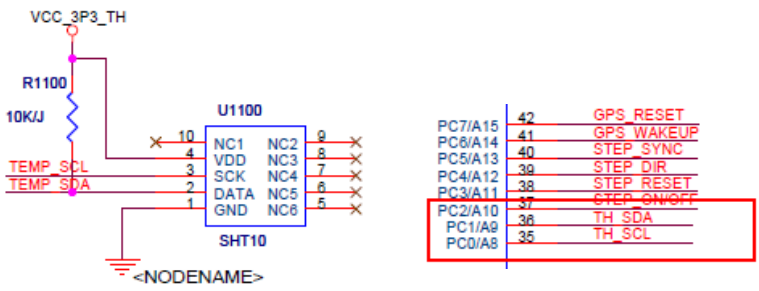
```
        stdout = &device;                       /* 표준출력으로 사용할 장치 설정 */

        GS_TwiWriteData(CTRL, 0x44);            /* Logic Low INT, 400kHz, Measurement mode */

        sei();                                  /* Global Interrupt Enable */

        while(1)
        {
                if(iFlag)
                {
                        iFlag = 0;

                        printf("X : %d\t", dataX);
                        printf("Y : %d\t", dataY);
                        printf("Z : %d\n", dataX);
                }
        }
        return 0;
}
/* ========================================================== */
```

실습 12. 온습도 센서 제어하기

12.1 온습도 센서(Humidity & Temperature Sensor) 이해하기

온습도 센서(HTS)는 하나의 데이터 핀과 하나의 클록 핀을 가지며, 기본적으로 GPIO만으로 제어가 가능하지만 동작 원리는 Two-wire Serial Interface로 I2C와 비슷하다. 단, ATmega128에서 지원하는 I2C 방식이 아니기 때문에 사용자가 직접 GPIO를 이용하여 신호를 만들어줘야 한다.

다음 그림은 온습도 센서(STH10)의 회로도와 실습 보드에 연결된 핀의 모습을 나타낸 것이다.



1. Power Pin

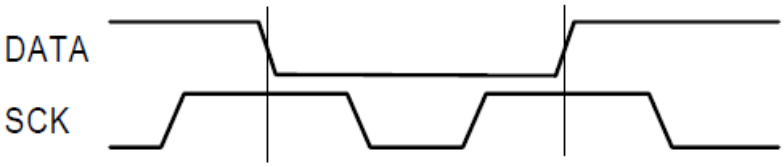
STH10은 Power-Up 후 Sleep mode에 진입하기까지 11ms가 소요된다. 따라서 처음 11ms 동안은 아무런 명령어도 전달하지 않도록 한다.

2. Serial Interface

STH10은 마이크로 컨트롤러와 STH10 사이의 동기화를 위한 Clock 신호 입력으로 SCK(SCL) 핀을 사용하며, Data의 입출력을 위해 DATA(SDA) 핀을 사용한다. DATA는 SCK 클록의 하강 에지(Falling Edge) 다음에 변화되며, 상승 에지(Rising Edge)에서 유효하게 된다. 데이터가 전송되는 동안 SCK는 High 상태를 유지해야 한다.

- Sending Command Sequence

통신 초기화를 위해 우선 MCU는 STH10으로 'Transmission Start' Sequence를 보내야 한다. Start 신호는 클록 SCK가 High일 때 DATA 신호가 High에서 Low로 변화해야 하며, SCK의 Low Pulse가 지난 뒤 다음 High Pulse에서 다시 High로 변화해야 한다. 다음 그림은 Start 신호의 Sequence를 나타낸 것이다.



Start 신호 전송 후 명령어를 전송하기 위해 3bit의 주소 값과 5bit의 Command Sequence를 전송한다. 3bit의 주소 값은 '000' 으로 고정되어 있으며, 5bit의 명령어는 다음과 같다.

Command	Code
Reserved	0000x
Measure Temperature	**00011**
Measure Humidity	**00101**
Read Status Register	00111
Write Status Register	00110
Reserved	0101x-1110x
Soft reset, resets the interface, clears the status register to default values wait minimum 11 ms before next command	**11110**

8번째 Clock의 하강 에지(Falling Edge) 후 STH10은 DATA 핀으로 Low 신호(ACK bit)를 출력하며, 9번째 Clock의 하강 에지 후 다시 High 신호로 돌아온다.

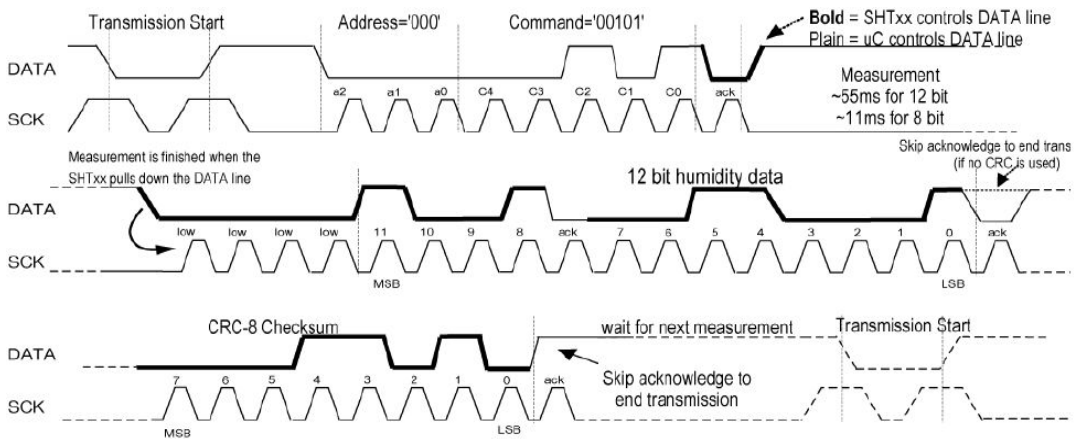
- **Measure Sequence**

MCU는 Measurement Command 전송 후 값이 측정되기를 기다려야 한다. 8/12/14 bit 측정값에 대해 11/55/210 ms 정도의 측정 시간이 소요된다. 값이 측정되는 동안 DATA 핀은 High로 유지되며 측정이 완료되면 STH10에 의해 Low로 변화한다. 단, SCK 핀은 DATA 핀이 Low로 변화할 때까지 Low 상태를 유지하여야 하며, 측정 완료 신호(DATA pulls down)가 감지되면 Clock을 발생시켜야 한다.

SCK의 Clock에 따라 2 바이트의 측정값과 1 바이트의 CRC Check-sum 값이 STH10으로부터 출력된다. MCU는 각 바이트에 대해 Low 신호로 ACK를 발생시켜야 한다. 모든 값은 MSB(최상위 비트)부터 전송된다. (12bit 값의 경우 5번째 Clock부터 bit 값이 출력되고, 8bit 값의 경우 첫 번째 바이트 값은 사용하지 않는다.)

CRC 바이트의 ACK 신호 후 통신이 종료되며, 만일 CRC Check-sum 값을 사용하지 않는 경우 측정값의 LSB(최하위 비트)의 ACK 신호를 High로 줌으로써 통신을 종료한다. 측정과 통신이 종료되면 STH10은 Sleep 모드로 자동 전환된다.

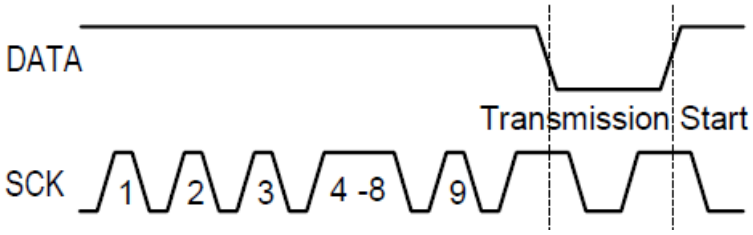
다음은 Serial Interface 동작 타이밍도의 예를 나타낸 것이다. Command 00000101b를 전송하고 12bit Humidity Data를 측정하여 수신 받는 과정으로, 굵은 선으로 표시된 부분이 STH10으로부터 출력되는 부분, 가는 선으로 표시된 부분이 MCU로부터 출력되는 부분을 나타낸 것이다.



- **Connection Reset Sequence**

MCU와 STH10 간의 통신에 문제가 생긴 경우 Reset Sequence를 통해 통신을 다시 시작할 수 있다. DATA 핀이 High로 유지되는 동안 SCK의 Clock이 9번 발생한 후 'Transmission Start' Sequence를 전송한 후 다시 Command Sequence를 진행할 수 있다. Reset Sequence는 Serial Interface만을 재설정하며, Status Register의 값은 재설정하지 않는다.

다음은 Reset Sequence의 타이밍도를 나타낸 것이다.



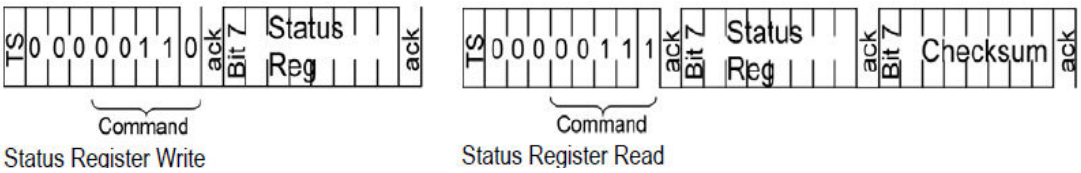
Serial Interface의 자세한 내용은 STH10의 데이터시트를 참조하도록 한다.

3. Status Register

STH10은 하나의 상태 레지스터를 가지고 있으며, 레지스터가 나타내는 각 비트의 값은 다음과 같다.

Bit	Type	Description	Default	
7		reserved	0	
6	R	End of Battery (low voltage detection) '0' for Vdd > 2.47 '1' for Vdd < 2.47	X	No default value, bit is only updated after a measurement
5		reserved	0	
4		reserved	0	
3		For Testing only, do not use	0	
2	R/W	Heater	0	off
1	R/W	no reload from OTP	0	reload
0	R/W	'1' = 8bit RH / 12bit Temperature resolution '0' = 12bit RH / 14bit Temperature resolution	0	12bit RH 14bit Temp.

Status Register의 값은 'Write Status Register'와 'Read Status Register' Command로 쓰거나 읽어 올 수 있으며 각 Command의 Sequence는 다음과 같다.



레지스터 설정 값에 대한 자세한 내용은 STH10의 데이터 시트를 참조한다.

12.2 온습도 센서 제어 실습하기

실습 1. 온습도 센서 제어 실습하기

1초마다 온습도를 측정하여 하이퍼터미널에 출력하라.

▣ HTS_Ctrl.c (Project name - HTS_Ctrl)

```
#include <avr/io.h>
#include <avr/delay.h>
#include <stdio.h>

/* ====================== STH 제어를 위한 선언 ===================== */
/* STH10 통신을 위한 매크로 정의 */
#define   HTS_DDR                 DDRC
#define   HTS_PORT                PORTC
#define   HTS_PIN                 PINC
#define   SCK_BIT                 0x01
#define   DATA_BIT                0x02

#define   SCK_SET_OUT             (HTS_DDR |= SCK_BIT)       /* SCK Output Mode */
#define   DATA_SET_OUT            (HTS_DDR |= DATA_BIT)      /* DATA Output Mode */
#define   DATA_SET_IN             (HTS_DDR &= ~DATA_BIT)     /* DATA Input Mode */

#define   SCK_HIGH                (HTS_PORT |= SCK_BIT)      /* SCK = 1 */
#define   SCK_LOW                 (HTS_PORT &= ~SCK_BIT)     /* SCK = 0 */
#define   DATA_HIGH               (HTS_PORT |= DATA_BIT)     /* DATA = 1 */
#define   DATA_LOW                (HTS_PORT &= ~DATA_BIT)    /* DATA = 0 */

#define   INPUT_DATA              (HTS_PIN & DATA_BIT)       /* DATA Input Value */
/* ============================================================ */

/* =============== Command Address  제어를 위한 선언 ================ */
/* STH10 Command Address 정의 */
#define   MEAS_TEMP               0x03            /* Measure Temperature */
#define   MEAS_HUM                0x05            /* Measure Humidity */
#define   READ_SR                 0x07            /* Read Status Register */
#define   WRITE_SR                0x06            /* Write Status Register */
#define   SOFT_RESET              0x1E            /* Soft Reset */
/* ============================================================ */

/* ===================== LED 제어를 위한 선언 ====================== */
/* LED 제어를 위한 매크로 정의 */
```

```
#define   LED_SET_OUT              (DDRE |= (1<<5))          /* LED1 Output Mode */
#define   LED_ON                   (PORTE |= (1<<5))         /* LED1 On */
#define   LED_OFF                  (PORTE &= ~(1<<5))        /* LED1 Off */
/* ============================================================ */

/* ====================== USART 제어를 위한 선언 ==================== */
#define   BAUD_RATE                51                        /* Baud Rate 19200 bps */
#define   DELAY_US                 1
/* ============================================================ */

/* ================= 온습도 값 저장을 위한 전역변수 선언 ================ */
/* 온습도 값 저장을 위한 전역변수 선언 */
volatile int gTempVal;
volatile int gHumVal;
/* ============================================================ */

/* ================= USART 표준 출력 위한 전역변수 선언 ================ */
static int USART0_putchar(char txData, FILE *stream);
static FILE device=FDEV_SETUP_STREAM(USART0_putchar, NULL, _FDEV_SETUP_WRITE);
/* ============================================================ */

/* ====================== USART 초기화 함수 구현  ================== */
/* USART0 레지스터 초기화 함수 구현 */
void HTS_InitUSART0()
{
        UBRR0H = (unsigned char)BAUD_RATE>>8;
        UBRR0L = (unsigned char)BAUD_RATE;

        UCSR0B = (1<<TXEN);                 /* Transmitter Enable */
        UCSR0C = (1<<UCSZ1) | (1<<UCSZ0);   /* No parity, 1 Stop bit, 8 bit Data */
}
/* ============================================================ */

/* ====================== USART 표준 출력 함수 구현  ================ */
/* USART0 Transmitter 함수 구현 */
/* 표준 출력으로 사용할 USART0_putchar 함수 구현 */
static int USART0_putchar(char txData, FILE *steam)
{
        /* New Line 출력의 경우 '\n\r' 로 바꿔줘야 함 (\n : new line, \r : carriage return) */
        if(txData == '\n')
```

```
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\n';
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = '\r';
        }
        else
        {
                while(!(UCSR0A & (1<<UDRE))) ;
                UDR0 = txData;
        }
        return 0;
}
/* ============================================================ */

/* ====================== Reset Sequence 함수 구현  ================ */
void HTS_ResetSequence()
{
        unsigned char i;

        DATA_SET_OUT;

        SCK_LOW;
        DATA_HIGH;
        _delay_us(10);

        for(i=0; i<9; i++)
        {
                SCK_HIGH;       _delay_us(DELAY_US);
                SCK_LOW;        _delay_us(DELAY_US);
        }
}
/* ============================================================ */

/* ============ Transmission Start Sequence 함수 구현  =============== */
void HTS_StartSequence()
{
        DATA_SET_OUT;

        SCK_LOW;
```

```
        DATA_HIGH;
        _delay_us(10);

        SCK_HIGH;       _delay_us(DELAY_US);
        DATA_LOW;       _delay_us(DELAY_US);

        SCK_LOW;        _delay_us(DELAY_US);

        SCK_HIGH;       _delay_us(DELAY_US);
        DATA_HIGH;      _delay_us(DELAY_US);

        SCK_LOW;
}
/* ============================================================ */

/* ============== Send Command Sequence 함수 구현  ================= */
unsigned char HTS_CommandSequence(unsigned char command)
{
        unsigned char i, ack;

        DATA_SET_OUT;

        SCK_LOW;
        DATA_HIGH;
        _delay_us(10);

        for(i=0; i<8; i++)
        {
                if(command & (0x80>>i))     DATA_HIGH;
                else                        DATA_LOW;

                SCK_HIGH;       _delay_us(DELAY_US);
                SCK_LOW;        _delay_us(DELAY_US);
        }

        /* Wait ACK Sequence */
        DATA_SET_IN;

        SCK_HIGH;
        ack = INPUT_DATA;
```

```
        _delay_us(DELAY_US);
        SCK_LOW;

        return ack;
}
/* ============================================================ */

/* =============== Measurement Sequence 함수 구현  ================= */
unsigned char HTS_ReadByteData()
{
        unsigned char i, rxData;

        DATA_SET_IN;

        for(i=0; i<8; i++)
        {
                SCK_HIGH;

                if(INPUT_DATA)    rxData = (rxData<<1) + 1;
                else              rxData = (rxData<<1);

                _delay_us(DELAY_US);
                SCK_LOW;
                _delay_us(DELAY_US);
        }
        return rxData;
}
/* ============================================================ */

/* ================= Write ACK Sequence 함수 구현  ================= */
void HTS_WriteACK(unsigned char ack)
{
        DATA_SET_OUT;

        if(ack)   DATA_HIGH;
        else      DATA_LOW;

        SCK_HIGH;         _delay_us(DELAY_US);
        SCK_LOW;          _delay_us(DELAY_US);
}
```

```
/* ============================================================ */

/* ========= Temperature & Humidity Calculation 함수 구현  ============ */
void HTS_CalculationHumTemp(int tempData, int humData)
{
        const float d1 = -40.0;
        const float d2 = 0.01;
        const float c1 = -4.0;
        const float c2 = 0.0405;
        const float c3 = -0.0000028;
        const float t1 = 0.01;
        const float t2 = 0.00008;
        int RH_linear;
        int RH_true;

        gTempVal = d1 + (d2*tempData);

        RH_linear = c1 + (c2*humData) + (c3*humData*humData);
        RH_true = (gTempVal-25) * (t1 + (t2*humData)) + RH_linear;

        gHumVal = RH_true;
        if(RH_true > 100)   gHumVal = 100;
        else if(RH_true < 0) gHumVal = 0;
}
/* ============================================================ */

/* ===================== Main 함수 구현  ========================== */
int main()
{
        unsigned int tempData, humData;
        unsigned char dataH, dataL;
        unsigned char ack;

        /* 레지스터 및 입출력 포트 초기화 */
        HTS_InitUSART0();
        LED_SET_OUT;                    /* LED Output Mode */
        SCK_SET_OUT;                    /* SCK Pin Output Mode */

        stdout = &device;               /* 표준출력으로 사용할 장치 설정 */
```

```
        _delay_ms(20);                          /* Wait for Device Sleep Mode */
        HTS_ResetSequence();                    /* Reset Sequence */

        while(1)
        {
                /* ==== Measurement Temperature Sequence ==== */
                HTS_StartSequence();                    /* Transmission Start Sequence */
                ack = HTS_CommandSequence(MEAS_TEMP); /* Send Command Sequence */

                _delay_us(10);

                DATA_SET_IN;                            /* Change DATA Input Mode */
                while(INPUT_DATA) ;                     /* Wait Complete Measurement */

                dataH = HTS_ReadByteData();             /* Read Measurement Data */
                HTS_WriteACK(0);                        /* Send ACK 0 */
                dataL = HTS_ReadByteData();             /* Read Measurement Data */
                HTS_WriteACK(1);                        /* Send ACK 1 (No Check-sum) */

                tempData = dataH & 0x3F;
                tempData = (tempData<<8) | dataL;
                /* ===================================== */

                _delay_ms(200);

                /* ===== Measurement Humidity Sequence ====== */
                HTS_StartSequence();                    /* Transmission Start Sequence */
                ack = HTS_CommandSequence(MEAS_HUM); /* Send Command Sequence */

                _delay_us(10);

                DATA_SET_IN;                            /* Change DATA Input Mode */
                while(INPUT_DATA) ;             /* Wait Complete Measurement */

                dataH = HTS_ReadByteData(); /* Read Measurement Data */
                HTS_WriteACK(0);                /* Send ACK 0 */
                dataL = HTS_ReadByteData(); /* Read Measurement Data */
                HTS_WriteACK(1);                /* Send ACK 1 (No Check-sum) */

                humData = dataH & 0x0F;
```

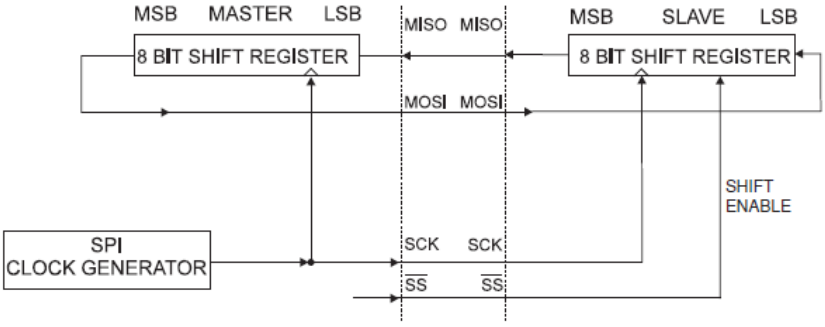
```
                humData = (humData<<8) | dataL;
                /* ===================================== */

                /* Calculation Humidity & Temperature */
                HTS_CalculationHumTemp(tempData,humData);

                /* Print Humidity & Temperature Data */
                printf("Temperature : %d, Humidity : %d\n", gTempVal, gHumVal);

                _delay_ms(1000);
        }
        return 0;
}
/* ========================================================== */
```

실습 13. SPI 제어하기

13.1 SPI(Serial Peripheral Interface) 이해하기

SPI(Serial Peripheral Interface)란 모토로라(Motorola) 사에서 제안한 시리얼 인터페이스 방식으로 SCK, MOSI, MISO의 3개 신호 선을 사용하고, 또 nSS(Slave Select) 신호선이 있어 슬레이브(Slave) 출력을 제어할 수 있다.

마스터와 슬레이브는 각각 Shift Register를 가지고 있어서, 마스터가 출력한 Clock에 동기화되어 Slave Shift Register를 1Byte 만큼 Shift 시키고 이 때 Slave는 Shift된 데이터를 읽는다. 다음 그림은 SPI에서 마스터와 슬레이브가 통신하는 과정을 간략하게 나타낸 것이다.



다음 표는 마스터와 슬레이브 모드일 경우 각각의 Pin에 대한 I/O 설정을 나타낸 것이다.

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

nSS Pin의 기능은 다음과 같다.

- Slave mode

SPI를 Slave로 설정하면 nSS Pin은 자동으로 Input mode가 설정된다. nSS Pin에 'Low' 가 입력되면 SPI가 활성화되고, MISO Pin이 Output mode일 경우 MISO로 데이터가 출력된다. SCK, MOSI Pin은 Input mode로 유지되어야 한다. nSS Pin에 'High' 가 입력되면 수신되는 데이터가 없다는 의미로 SPI Logic을 Reset 시킨다. 만약 데이터를 수신하는 동안 nSS Pin에 'High' 가 입력되면 SPI는 즉시 수신을 중지하게 되고 데이터를 손실하게 된다.

- Master mode

SPCR 레지스터의 MSTR bit를 Set하여 마스터 모드로 설정하게 되면, 사용자는 nSS Pin을 Output mode로 설정하여야 한다. 만약 nSS Pin이 Input mode로 설정되어 있을 경우, 장치가 Master로 동작하기 위해서는 nSS Pin에 'High' 를 입력해야 하며, nSS Pin에 'Low' 가 입력될 경우 SPI 시스템은 Master mode를 해제시킨다.

13.2 SPI 레지스터

다음은 SPI 제어에 사용되는 레지스터에 대한 설명이다.

1. SPCR (SPI Control Register)

SPI의 송수신을 제어한다.

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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	

- **SPIE** (SPI Interrupt Enable) : SPI 인터럽트를 허용한다.
- **SPE** (SPI Enable) : SPI를 활성화 시킨다.
- **DORD** (Data Order) : 이 bit가 '1' 로 설정되면 Data를 전송할 때 LSB(Least Significant Bit)부터 전송된다.
- **MSTR** (Master/Slave Select) : Master/Slave mode를 설정한다. '1' 로 설정될 경우 Master mode로 동작하며, '0' 일 경우 Slave mode로 동작한다.

2. SPSR (SPI Status Register)

SPI의 송수신 상태를 제어한다.

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **SPIF** (SPI Interrupt Flag) : SPI 전송이 완료되면 '1' 로 Set 된다. SPCR의 SPIE bit가 설정되어 있을 경우 인터럽트를 발생시키며, 인터럽트 처리가 완료되면 '0' 으로 Clear 된다.

SPI가 Master mode이고 nSS Pin이 Input mode인 경우, nSS Pin에 'Low' 가 입력되면 '1' 로 Set 되며, SPSR을 처음 읽거나 SPDR을 읽으면 '0' 으로 Clear 된다.

- **WCOL** (Write Collision Flag) : SPI가 데이터를 전송하는 동안 SPDR 레지스터에 값을 쓰면 이 bit가 '1' 로 Set 된다. SPSR을 처음 읽거나 SPDR을 읽으면 '0' 으로 Clear 된다.

3. SPDR (SPI Data Register)

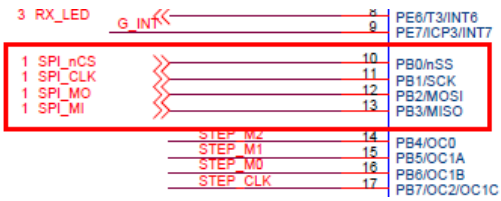
SPI의 송수신 레지스터로, 읽어올 때는 수신 데이터 레지스터, 쓸 때는 송신 데이터 레지스터로 동작한다.

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

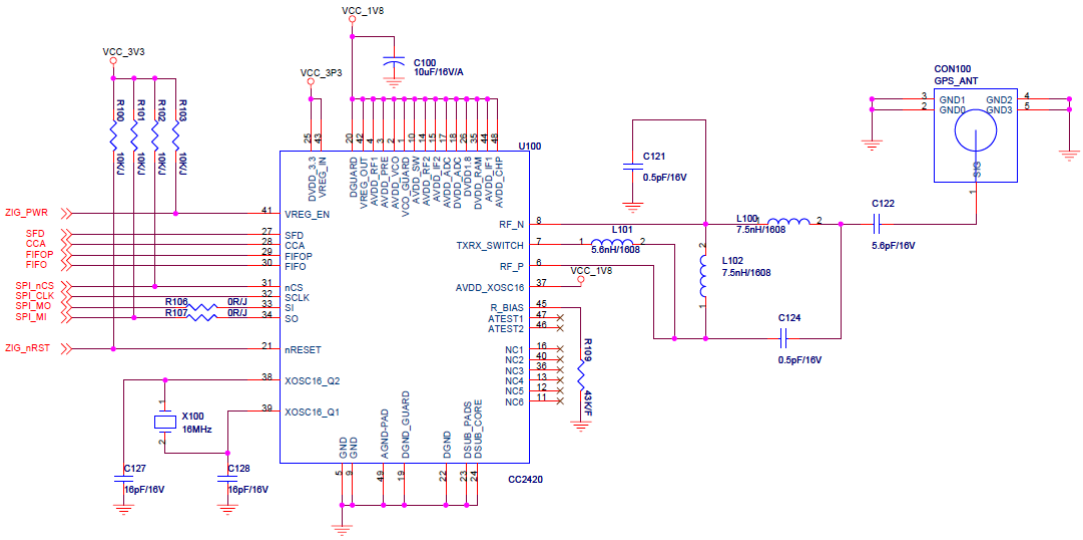
자세한 내용은 ATmega128의 데이터 시트를 참조한다.

13.3 SPI 제어 실습하기

다음 그림은 실습 보드에 연결된 SPI Pin을 나타낸 것이다.



현재 실습 보드에서 SPI는 서버 메인과 Zigbee 통신을 하기 위해 사용되고 있다. 다음은 Zigbee 모듈의 회로도를 나타낸 것이다.



Zigbee 통신에 대한 실습 예제는 CD에 수록되어 있는 소스를 참조하도록 한다.