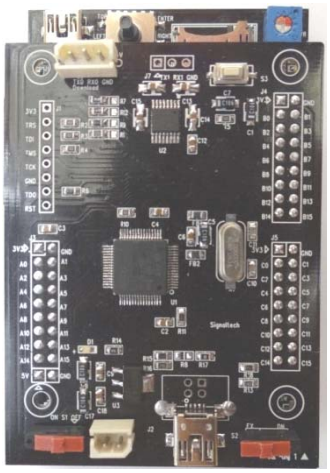
ARM Cortex Processor

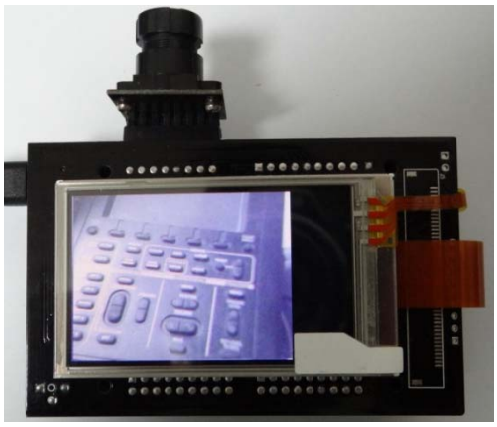
# STM32F103 응용

SignalTech

Cortex Processor

# STM32F103 응용





SignalTech

# 목차

# 서 론

STM32 ARM 사에서 새로운 코어가 Cortex다. ARM7, ARM9 많이 사용되었는데, 그 다음ARM11 대신 그를 초과하는 내부 코어를 ARMv5로 교체하고 많은 부분을 개선한 후 새로운 이름 "**Cortex**" 새로 사용되고 있다.

ARM7 과 다르게 Cortex M processor 는 32bit CPU, 버스 아키텍처, 중첩된 인터럽트 유닛, 디버그 시스템, 표준 메모리 레이아웃을 포함하는 표준화 된 마이크로 컨트롤러이다. Cortex-M3 는 하버드 아키텍쳐이며, 그래서 병렬작업과 전반적인 성능 향상을 위해 다중 버스를 가진다. 이전의 ARM 아키텍처와 달리 Coretex 제품군은 unaligned 데이터 액세스를 허용한다. 이는 내부 SRAM 사용에 최고의 효율성을 보장한다

.

비록 Cortex 저가 코어로 설계되었지만, 그것은 여전히 32 비트 CPU 와 같은 두 가지 작동 모드-스택을 구성할 수 있는 스레드 모드와 핸들러 모드-에 대한 지원을 하고 있다. 이는 보다 정교한 소프트웨어 설계 및 Real-Time 운영 체제에 대한 지원을 한다. Cortex 코어는 또한 RTOS 가 커널에 대한 인터럽트를 제공하기 위해 24 비트 오토 리로드 타이머를 포함한다. ARM7/ ARM9 CPU 들이 두 가지 명령 셀 (ARM 32 비트와 Thumb 16 비트 명령 셀)을 가지는 반면, Cortex 제품군은 ARM Thumb-2 명령 셀을 지원하도록 설계되었다. 이는 ARM32 비트 명령 셀의 성능을 물려받고, Thumb16 비트 명령 셀의 코드밀도로 셋팅하기 위하여 양쪽 모두와 조화를 이룬다. Thumb-2 명령 셀은 C/C++컴파일러를 위한 타겟으로 설계된 매우 풍부한 명령 셀이다. 이는 Cortex 어플리케이션이 C 로 완전한 코딩이 될 수

있음을 의미한다. STM32 는 14 가지 다른 variants 로 처음 공개되었다. 이는 CPU clock 을 72MHz 의 상향한 성능 라인과 최대 36MHz 로 작동하는 액세스 라인의 두 그룹으로 나뉘어진다. 성능 라인으로 분할된다. 다른 CPU 보다 같은 핀 수에 비해 더 많은 기능, 뛰어난 처리 속도, 저렴한 가격, 훨씬 쉬워진 다운로드는 시리얼 통신으로 가능해져 월등하다. 여기서는 STM32F103RCT6, STM32F103RET6 를 테스트 하지만, 다른 CPU 하고도 프로그램 호환성이 있다. 테스트는 서브 보드 부착되어 것을 기준으로 설명했다. 메인보드에 64 핀 STM32F103R8T6, STM32F103RBT6 도 가능하다. STM32 를 단계별 사용절차를 정리 하였다.

## < STM32 사용 단계 순서 >

### 1. Keil 설치 및 사용 순서

1단계 : 컴파일러 설치 (STM32관련 디바이스를 컴파일 하기 위해 필요함)

Stm32를 컴파일러 프로그램 설치 (여기서는 Keil 사용 7 페이지 참고)

2단계 : 다운로드 프로그램 설치. 1단계에서 생성된 파일을 .HEX 이나 .BIN 을 다운로드 하기 위한 프로그램을 설치한다. 아래 1) 이나 2)중 하드웨어가 준비된 것 중 하나를 선택해 프로그램을 설치한다.

1) ST 사의 Flash Loder Demo(20 페이지 참고) => 권장사항

2) H-JTAG 패럿포트(23 페이지 참고)

3단계 : 다운로드 디바이스 설치

1) PC 시리얼포트인 경우 설치 안하고 바로 COM 1, 2 사용해서 다운로드

1단계 -> 2단계 1)을 진행을 한다.

2) 시그널테크 USB232 사용할 경우 Prolific사의 PL2303 Installer Driver 설치

1단계 -> 3단계 -> 2단계 1)을 진행을 한다.

3) H-JTAG 패럿포트

1단계 -> 2단계 2)를 진행을 한다.

4단계 : 다운로드하고 메인보드의 S2 스위치를 실행(EX)으로 위치하고 리셋 버튼을 누르면 실행한다. (37페이지 참고)

## 2. CooCox CoIDE 설치 순서

1) GCC는 Toolchain **arm-none-eabi-gcc-4_6-20111208.exe** 설치
2) CooCox 설치를 관리하는 CoCenter-1.4.9.exe 파일을 실행(업데이트로 지금 버전과 다를 수 있음)
3) CoCenter-1.4.9.exe 내의 프로그램을 하나씩 실행(필요한 부분만 설치하나 권장사항은 전부 설치한다.)
4) 설치하고 나면 바탕화면에 CoIDE 를 실행해서 프로그램 작성하고 다운로드하고 실행한다.

무료 개발 툴인 CoIDE는 매우 편리한 코드편집과 디버깅환경을 제공한다.

# 제1장 프로그램 개발 환경

ARM 컴파일에는 여러 종류가 있으나, Keil 사의 RealView 에는 하드에어가 없어도 미리 결과를 알 수 있는 시뮬레이션기능이 있는 Keil사 μvision 이다.
STM32F4을 사용하고자 하면 V4.0 이상 버전을 사용하여야 한다. 하드웨어 디바이스가 지원되지 않기 때문이다.
다음절에 소개할 **CooCox CoIDE**는 다른 무료컴파일에 비해 제약이 없고, 사용하기에 편한 것 같다. 여기서는 두 가지 프로그램에서 간단히 정리 해 놓았다.
CooCox IDE는 오픈 소스인 GNU GCC는 Eclipse 기반의 프로그램을 설치한다. GNU GCC 컴파일러에 의존하기 때문에 설치를 해야 한다. AVR를 사용 해 보았다면, AvrStudio 설치 하기 전 Win Avr를 설치하는 것과 같다. CooCox IDE가 AvrStudio 이라고 생각하면 된다.
최근 프로그램을 다운 받아 업데이트 설치하면 STM32F4 지원을 추가되어 있다.
또한 STM32F4Discovery보드를 지원할 수 디바이스를 지원한다.
CooCox IDE는CMSIS 및 대한 부팅 파일을 포함할 수 있으므로, 자체적으로 프로젝트를 생성 기능과 함께 예제 프로젝트가 실제로 코드를 변경되어 작성하기에 쉽다.
다른 컴파일처럼 환경설정이 복잡하지 않다. AVR 프로그램의 Codevision 처럼 프로그램이 자동 생성된다.

## 1.1 Keil 프로그램 설치

Keil 사의 ARM 컴파일 툴(RealView)의 μvision V4.0 을 설치한다. 여기서는 버전4.0을 설치를 보여주고 있으나 4.20 이상을 권장한다. 버전4.20 이상을 설치하면 STM32F4Discovery보드를 지원할 수 디바이스를 지원하고, 프로그램 내에서 ST-Link를 사용하여 프로그램 다운로드를 사용할 수 있다. STM32F437,439 는 버전 5.0 이상이어야 한다.
[그림1.1.1] 설치프로그램에서 MDK400.exe 더블 클릭하여 실행한다.

[그림1.1.1] 설치 프로그램 파일
Next

[그림1.1.2] 프로그램 설치

동의에 체크를 하고 Next

[그림1.1.3] 동의 체크

Next

[그림1.1.4] 설치 디렉토리

기록하고 Next

[그림1.1.5] 사용자 등록

Next




[그림1.1.6] 설치

Finish 클릭

[그림1.1.7] Finish

일단 프로그램 설치는 끝났다. License 키 등록을 한다.

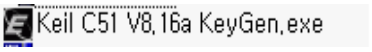
- **License 등록**

License 키를 입력하기 위해 Keil μvision 프로그램을 실행한다. [그림1.1.8] 처럼 File =>License Management 를 클릭한다.

[그림1.1.8] License Management

아래 [그림1.1.9]의 CID값을 통해 LIC 값을 넣어야 한다.

[그림1.1.9] CID



설치 프로그램내 Keil C51 V8.16a KeyGen.exe 실행한다. 위 [그림1.1.9] 의 CID값을 넣는다. Target는 ARM 를 선택하고 아래는 Prof. Developers Kit/Realview MDK 를 선택하고, Genrate 를 클릭하면 아래 [그림1.1.10] 처럼 LIC 가 나오고 이를 아래 왼쪽 [그림1.1.10] 처럼 복사 해 넣으면 된다. 아래 창에 성공했다는 메시지가 나오면 설치가 끝났다.

[그림1.1.10] LIC 등록

**Win 7인 경우 실행이 안될 경우 다음 아래 사항을 참고 한다.**

< Win 7 OS 인 경우에만 해당>
라이센스 등록하기 전에 윈도우XP 모드에 놓고 실행한다. 그러기 위해서는 설치 후 실행 파일에 커서를 위치하고 마우스 오른쪽 버튼을 누르면 아래 화면이 나온다. 속성을 선택한다.

호환성 메뉴에 위치하고, 호환모드는 아래 그림처럼 체크한다.

## 1.2 Keil 환경설정

[그림1.2.1]은 필요한 기본 디렉토리와 파일이다. μvision 실행전 준비한다.

[그림1.2.1] 기본 디렉토리, 파일

프로젝트 디렉토리를 만든다. 1_GPIO_LED 다음으로 Libraries 만들고, 그 안에 CMSIS과 STM32F10x_StdPeriph_Driver 디렉토리를 만든다. 그리고, 프로젝트 디렉토리 내에 Startup, test, USER 만든다. 기본적으로 필요한 파일이므로 복사해서 사용한다. 그리고 프로젝트 파일을 만들기 위해 Keil μvision 실행한다.
[그림1.2.2]은 Keil μvision 프로그램을 실행한 화면이다.



[그림1.2.2] 프로그램 실행 화면

[그림1.2.3]처럼 Project ->New uVision Project 프로젝트 명이 생성한다.

[그림1.2.3] New uVision Project 프로젝트 생성

라이브러리등 필요한 파일을 등록해야 한다. 프로젝트 파일면에 마우스를 대고 오른쪽 버튼을 누르거나 아래 [그림1.2.4] 처럼 박스내를 클릭한다.

[그림1.2.4] 라이브러리등 파일 등록

아래 [그림1.2.5]처럼 등록한다.

[그림1.2.5] 라이브러리등 파일 등록

아래 [그림1.2.6]은 등록된 파일이다.

[그림1.2.6] 파일 등록한 프로젝트

[그림1.2.7] 각종 옵션 등을 설정 하는 곳이다.

[그림1.2.7] 옵션 등록

[그림1.2.8] 디바이스 선정한다.

[그림1.2.8] 디바이스 선정

[그림1.2.9] 처럼 옵션 중 Include Paths 내에는 파일 위치에 맞게 지정해 주어야 한다.
..\Libraries\CMSIS;..\Startup;..\Libraries\STM32F10x_StdPeriph_Driver\inc;..\Libraries\STM32F10x_StdPeriph_Driver\src;..\USER

[그림1.2.9] Include 등록

지금까지 설정이 어려우면 제공한 프로그램 파일을 복사해서 main 부분과 필요한 부분을 수정해서 사용한다.

참고

- CMSIS (Cortex Microcontroller Software Interface Standard) 타 회사 칩을 사용하더라고 firmware 의 호환성을 높이기 위해 제공.
- DfuSe 용도는 USB 포트로 다운로드용도이다. 제품 출하 후 사용자가 프로그램 업데이트 할 수 있게끔 제공한 것이다.
- USB 를 사용할 때 D+를 Low 상태(USB 연결 안된 상태) =>초기화 상태 명령이 끝난 후 => D+ 풀 업이 되어야 인식이 된다.
- 여기서 사용하는 cpu는 startup_stm32f10x_hd.s 사용

[그림1.2.10] STM32F103 계열

[그림1.2.10] 에서 보면 STM32F103계열에 따라 Startup 이 바뀐다.

1) startup_stm32f10x_ld.s
2) startup_stm32f10x_md.s
3) startup_stm32f10x_hd.s

여기서는 startup_stm32f10x_hd.s사용한다. 다른디바이스를 사용할때는 위 STM32F103 계열을 참고한다.

## 1.3 CooCox 설치

GCC 툴을 설치하고, CooCox를 설치한다. CooCox를 처음 접했을 경우 모든 프로그램을 회원 가입이 없이 다운 받았는데, 요즘은 그렇지 않는 것 같다. 그리고 업데이트도 자주하고 있다.

오픈소스 GCC 환경을 제공하는 사이트에 알아보면
1) Yagarto ; http://www.yagarto.de/
2) WinARM ; http://gandalf.arubi.uni-kl.de/avr_projects/arm_projects/index_cortex.html
3) GNUARM ; http://www.gnuarm.com/

이들 중 최신 디바이스를 지원하는 Yagarto 의 GCC 툴을 사용한다
WinARM, GNUARM 는 최신 버전이 업데이트 되어 있는지 확인하고 사용한다

무료 개발 툴인 CoIDE와 오픈소스 GCC를 이용하는데, 이클립스 기반이므로 매우 편리한 코드편집과 디버깅환경을 제공한다.
다른 무료 개발 툴들은 프로그램 사이즈가 제한되어 있고, 등록하라는 메시지가 번거로운것에 비해 사용하기에 좋다.

CooCox CoIDE 설치순서
1) GCC는 Toolchain **arm-none-eabi-gcc-4_6-20111208.exe** 설치
2) CooCox 설치를 관리하는 CoCenter-1.4.7.exe 파일을 실행
3) CoCenter-1.4.7.exe 내의 프로그램을 하나씩 실행(필요한 부분만 설치한다. CoSmat는 디바이스 지원이 거의 없기 때문에 설치 시 제외 해도 좋다.) 설치하고 나면 바탕화면에 CoIDE 를 실행해서 프로그램 작성하고 다운로드하고 실행한다.

### 1) GCC 설치

GCC는 Toolchain 설치하기 위해서는 아래 홈페이지에 들어가 프로그램을 다운로드 받는다.
https://launchpad.net/gcc-arm-embedded/4.6/2011-q4-major
아래는 홈페이지의 일부분이다. 이중에서 " **arm-none-eabi-gcc-4_6-20111208.exe** (md5) Windows installer " 윈도우 설치용을 다운받는다.

| | |
|---|---|
| **arm-none-eabi-gcc-4_6-20111208.tar.bz2** (md5) | Linux installation tarball |
| **arm-none-eabi-gcc-4_6-20111208.exe** (md5) | Windows installer |
| **arm-none-eabi-gcc-4_6-20111208-src.7z.002** (md5) | Source package part 2/2. Use 7z to extract. |
| **arm-none-eabi-gcc-4_6-20111208-src.7z.001** (md5) | Source package part 1/2. Use 7z to extract. |

다운 받은 **arm-none-eabi-gcc-4_6-20111208.exe**를 실행한다. 아래 화면이 나오면 선택을 하고 OK 를 클릭한다.

[그림1.3.1] 설치 시작 언어 선택

Yes 를 선택하고 클릭한다.

[그림1.3.2] 설치 여부 확인

Next를 클릭한다.




[그림1.3.3] Install 화면

동의에 체크를 하고, Next를 클릭한다.

[그림1.3.4] 설치동의 확인

Next를 클릭한다.

[그림1.3.5] 설치 진행

Next를 클릭한다.

[그림1.3.6] 설치Type 선택

Next를 클릭하면 인스톨을 한다..

[그림1.3.7] 설치 진행

아래 화면처럼 Finish 를 클릭하여 설치를 끝낸다.

[그림1.3.8] 설치 종료

아래화면이 나오는데 오른쪽 상단 x 를 클릭하여 빠져나온다.

[그림1.3.9] 설치 마지막 화면

GCC는 Toolchain 설치가 마무리 되었다.

### 2) CooCenter 설치

CooCox 설치를 관리하는 CoCenter-1.4.7.exe 파일을 실행파일을 다운 받기 위해 아래 홈페이지에 들어간다.

http://www.coocox.org/CoLinkGuide/CoMDKPlugin_Updates.htm
아래 부분은 홈페이지 상단 부분이며, 최신 버전을 받을 수 있다. 아래는 최근 버전이며, 아래는 디바이스 추가부분을 보여준다.
ST , **STM32F4DISCOVERY 07IG** 사용 하기 위해서는 이 버전을 받아야 한다.

Download the latest CoMDKPlugin
**CoMDKPlugin V1.4.2** 5/11/2012

**[CoMDKPlugin-Improved]**

- Added: Supported Cortex M4 (FPU Register included)
- Added: Supported ST:**STM32F405RG, STM32F407IG, STM32F407VG, STM32F407ZG, STM32F415RG, STM32F417VE**
- Added: Supported NXP: LPC1100, LPC1111 LPC1112, LPC1114, LPC1115, LPC11C22, LPC11C24, LPC11U23, LPC11U24, LPC1315, LPC1316, LPC1317, LPC1345, LPC1346, LPC1347
- Added: Supported Holtek: HT32F1755, HT32F1765, HT32F2755

Download the latest CoMDKPlugin 를 클릭하면 CoCenter-1.4.7.exe를 다운로드를 받는다.

CoCenter-1.4.7.exe 실행 파일을 실행한다.
Next를 클릭한다.

[그림1.3.10] Setup 화면

Next를 클릭한다.

[그림1.3.11] 설치디렉토리 설정

Install을 클릭한다.

[그림1.3.12] 설치 진행

아래 화면 처럼 Finish 를 클릭하여 설치를 끝낸다.

[그림1.3.13] 설치 종료 화면

CoCenter 내의 프로그램을 하나씩 실행하여 설치한다.

[그림1.3.14] CoCenter 설치화면

계속 선택하여 프로그램을 설치한다. 첫 번째 CoIDE로 컴파일러이므로 필히 설치해야 한다. CoSmat 는 디바이스 지원이 거의 없기 때문에 설치 시 제외 해도 좋다. 또한 나머지도 필요 시 프로그램만 설치한다.

아래 그림은 설치 후 화면이다.

[그림1.3.15] CoCenter 설치후 화면

## 1.4 CoIDE 환경설정

CoIDE 를 실행한다

[그림1.4.1] CoIDE 실행화면

프로그램은 시작은 기존 프로그램은 Open a Project를 선택하고 새로운 프로젝트는 Create a New Project 을 클릭하여 선택한다.

[그림1.4.2] 프로젝트 이름과 경로

프로젝트 이름과 경로를 설정한다. 위그림은 예를 들어 test로 하였다.

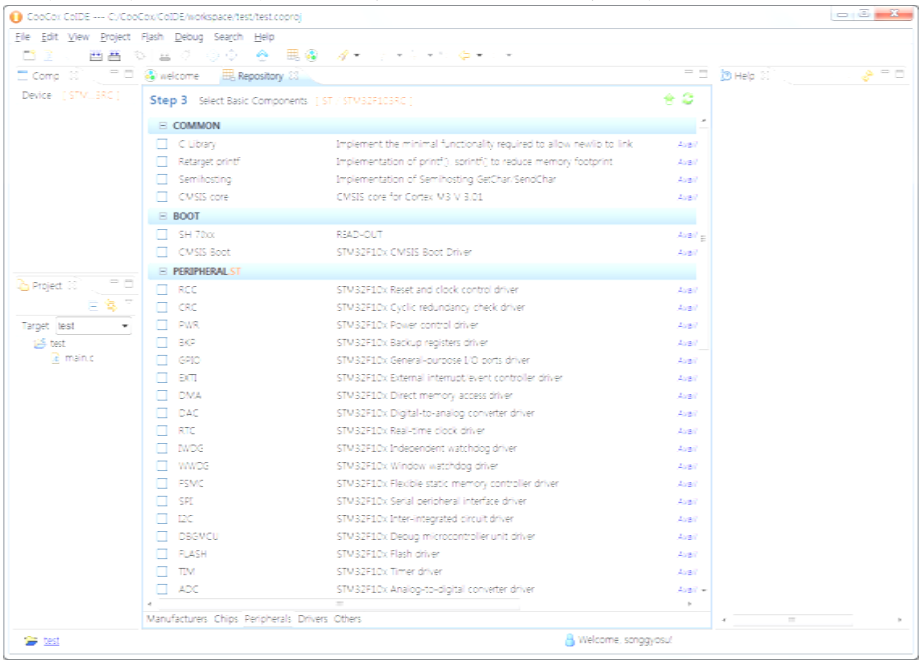
아래그림은 디바이스는 Chip를 CooCox에서 지정한 보드는 Board를 하는데 여기서는 일반 디바이스를 가지고 하기 때문에 Chip을 클릭하고, Next를 클릭한다.

[그림1.4.3] 디바이스 또는 보드 설정

여기서는 디바이스 제조사 선택하고, 테스트할 보드의 디바이스를 보고 STM32F103RC 또는 STM32F103RE 를 선택한다. 아래는 STM32F103RC를 테스트 한다.

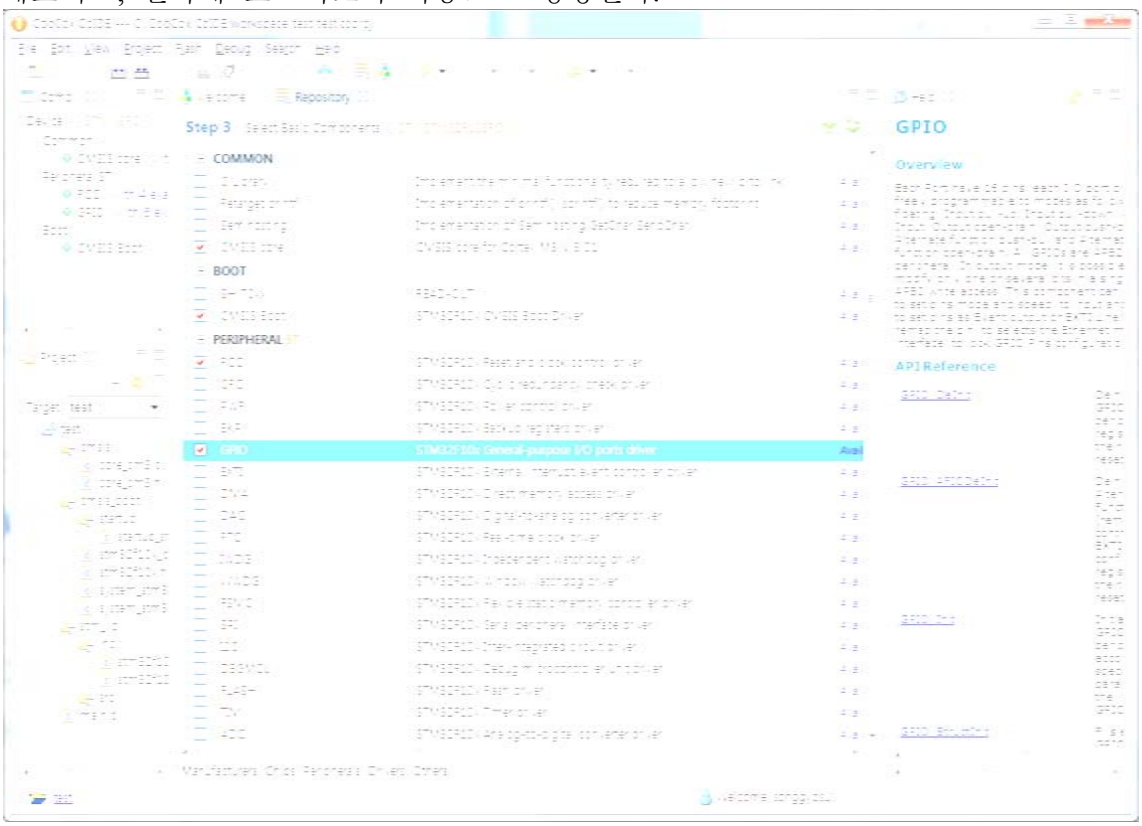
[그림1.4.4] 제조사와 디바이스 설정

Finish를 클릭하면 아래그림이 나온다. 아래 부분은 테스트하고자 하는 구성을 선택한다. 만약 I/O를 테스트할려면 GPIO를 선택한다.



[그림1.4.5] 기본 구성 설정

만약 I/O를 테스트할려면 GPIO를 선택한다. 그러면 GPIO 에 관련된 것은 자동으로 체크되고, 왼쪽에 프로젝트가 자동으로 생성된다.



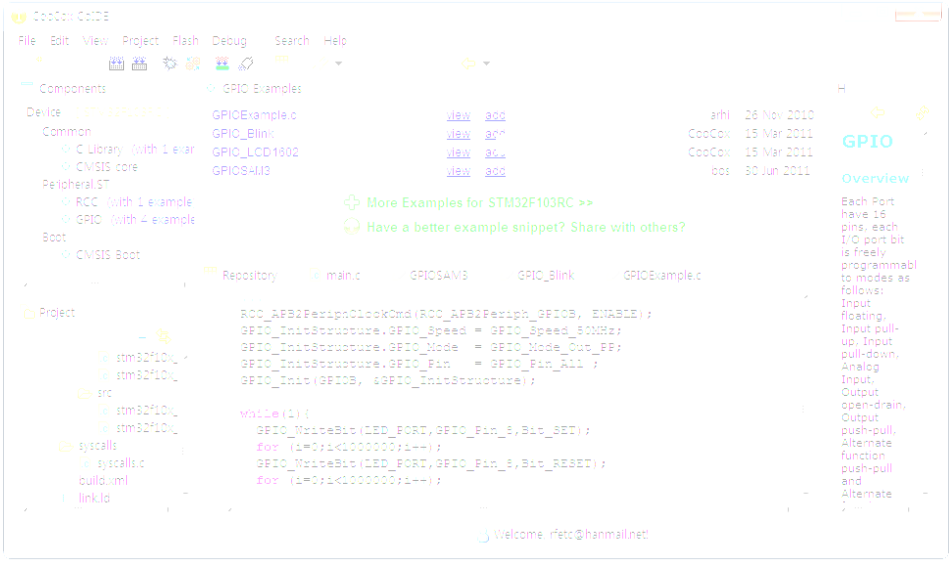
[그림1.4.5] GPIO 설정

왼쪽에 main.c 를 클릭하면 아래 화면처럼 main 화면이 생성된다.



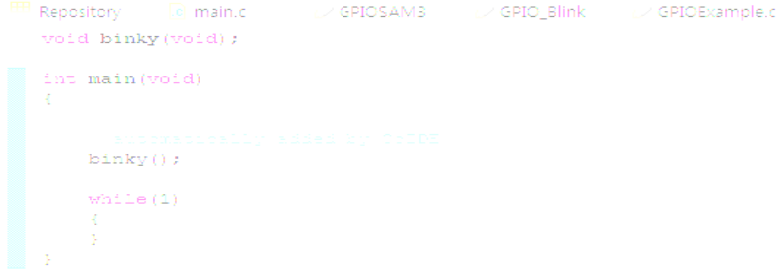
[그림1.4.6] CoIDE main 화면 생성

여기서 코딩하여 넣어도 되지만, 편리한 기능을 사용해 본다. 왼쪽 상단에 Components 아래에 GPIO (with 7 examples) 를 클릭한다.
아래 그림처럼 GPIO Examples 프로그램이 4가지 있다. 이중에서 원하는 프로그램에 근사한지 view 를 클릭하여 보고 add를 클릭하여 프로그램을 자동으로 붙여 넣을 수 있다.



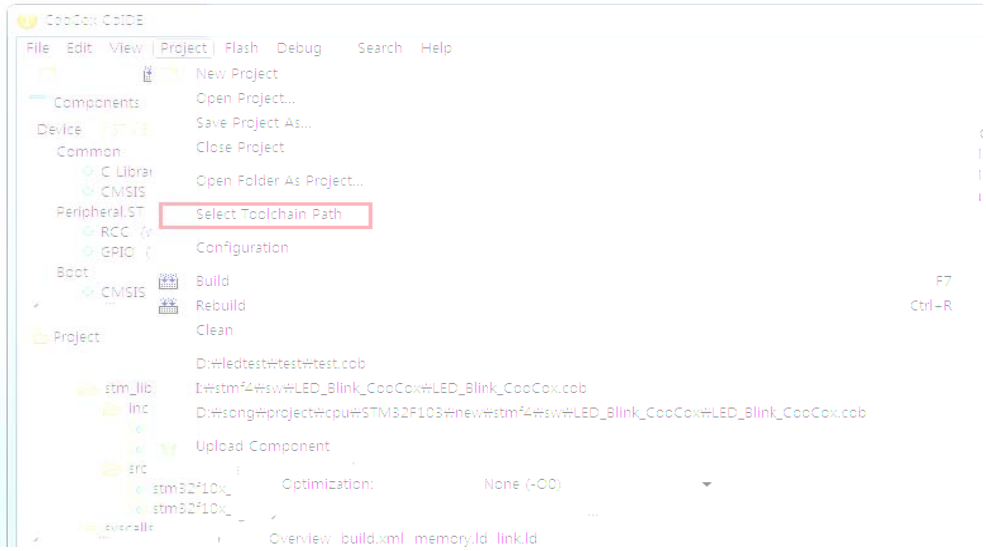
[그림1.4.7] CoIDE 프로그램 자동 코딩생성

프로그램이 추가되어 있는 것을 알 수 있다.



[그림1.4.8] CoIDE main 프로그램 확인

컴파일 옵션을 지정한다.



[그림1.4.9] CoIDE 컴파일 옵션 지정

아래 그림처럼 지정을 한다. 그리고 OK버튼을 클릭한다.
(**arm-none-eabi-gcc-4_6-20111208.exe** 를 설치한 경로이다)

[그림1.4.10] CoIDE 디렉토리 지정

여기서 뒤 장에 나오는 실습에 서브보드 없이 LED를 구동 할려면 컴파일 해서 다운로드 한 다음 결과를 확인 할 수 있다.

여기서 뒤 장에 나오는 실습에 있는 서브보드에 LED를 구동하기 위해 수정을 한다. Binky() 더블 클릭하여 F3을 누르면 void binky(void) 함수로 이동한다.

[그림1.4.11] CoIDE 프로그램 수정

수정 후 아래 그림처럼 Rebuild 하면 컴파일이 실행된다.

[그림1.4.12] CoIDE 프로그램 컴파일

Test=> Debug =>bin 아래 실행파일(test .hex)을 다운로드 하여 실행한다.

[그림1.4.13] CoIDE 컴파일 결과 확인

## 1.5 프로그램 다운로드

이절에서는 프로그램 커파일 후 생성 된 파일을 디바이스에 다운로드 방법에 대해 알아본다. 여기서는 Serial (USB) , ST-LINK/V2, H-JTAG 3가지 방법을 설명한다.

### 1.5.1 Serial,USB

**[그림 1.5.1] 하드웨어 다운로드설정**

[그림 1.5.1] 다운로드에서 1) 시리얼 포트를 사용할 경우는 컴퓨터의 시리얼 포트를 사용할 경우 이다. 2) USB 포트를 사용할 경우는 PL2303 드라이브 프로그램을 설치해야 한다. 만약 이 프로그램이 설치 되어 있으면 설치하지 않아도 된다. USB 프로그램을 다운 받아야 한다. PL2303 제조사인 http://www.prolific.com.tw 에서 프로그램을 다운로드 받을 수 있다. 다운 받아 압축을 푼 후 설치한다. 실행하면 아래 그림이 나온다. Next 를 클릭한다.

[그림 1.5.2] USB 프로그램 설치

실행하면 아래 그림이 나온다. Finish 를 클릭하여 설치가 끝난다.

[그림 1.5.3] USB 프로그램 설치 완료

보드에 USB 커넥터를 연결하고 PC 윈도우 화면의 시작-> 설정->제어판->시스템 ->하드웨어 ->장치관리자 ->포트 (COM 및 LPT) -> Prolific USB-to- Serial Comm Port (COM3)을 확인한다.

[그림 1.5.4] USB 프로그램 설치 확인

Prolific USB-to- Serial Comm Port (COM3) 이 설치 되어 있는데 설치 할 때 COM3 로 항상 나오는 것은 아니다. Com 포트가 이상이 있을 때 다운로드가 안될 때 여기를 꼭 보고 한다. COM3 를 다른 포트로 변경할 때는 Prolific USB-to- Serial Comm Port (COM3) 더블 클릭한다. 포트설정에 들어가 들어간다.

[그림 1.5.5] 포트 설정

아래 [그림 1.5.6] 고급 설정에 들어가 사용하고자 하는 포트를 선택하면 된다. 단 COM1,2 는 사용이 안 된다. 컴퓨터 Bios 에서 COM1,2 사용하기 때문이다. 요즘은 COM1,2 포트가 없는 PC 도 있다. 주의하자! USB 로는 COM1,2 는 사용하지 말자. 또, 사용이 안 되는 포트는 현재 이미 사용 중이어서 안 된다.

[그림 1.5.6] 고급 설정

ST홈페이지에 들어가서 Manual 에서 다운로드 프로그램이 있는 UM0462 의 파일을 다운받는다.

사이트 가서(Design Resource 로 이동해 아래) 다운로드 프로그램 받기 (아래 그림은 참고*홈페이지 업데이트를 자주 하므로 실제 화면과 다를 수 있다.)

[그림 1.5.7] ST홈페이지에 UM0462 의 파일

압축을 풀어 Flash_Loader_Demonstrator_v2.6.0_Setup.exe 을 클릭하여 설치 한 다음 Flash Loder Demo를 클릭한다.


[그림 1.5.8] Flash Loder Demo 실행파일

1. 메인보드 S1 전원스위치를 ON 하고, S2 를 DN 에 놓으면 다운로드하기 위해 준비가 되었다. 아래 그림처럼 설정한다. COM 포트에 맞게 설정한다. 그리고 Next 를 한다.

[그림 1.5.9] 환경설정

2. ARM 보드가 제대로 연결 되면 아래 [그림 1.5.10] 처럼 진행된다. 오른쪽 그림은 연결 된 디바이스에 정보를 표시 해 주고 있다. 이상이 없으면 Next 를 한다.

[그림 1.5.10] 설치

3. 그리고 아래 [그림 1.5.11] 그림처럼 .bin 또는 .hex 파일을 선택한다. 다음 Next 를 한다.

1) Erase 에서 All 은 전체 프로그램 삭제, Selection 는 선택 삭제 2) Jump to the user program 을 체크하면 선택하면 자동으로 부트 모드에서 프로그램 실행한다. ( S2 를 DN 에 놓은 상태에서 실행 가능, 다시 다운로드하기 위해서는 리셋 후 다운로드 가능하다.) Verify after download 는 검사 후 다운로드를 한다.

[그림 1.5.11] 다운로드 옵션

4. [그림 1.5.12]은 프로그램을 다운로드 완료 한 상태이다.

[그림 1.5.12] 다운로드 완료

다운로드 모드에서 실행모드로 전환 한 다음 리셋 버튼을 누르면 실행된다. 다시 프로그램을 다운로드를 할려면 리셋 버튼을 누르고 다시 다운로드 하면 된다.

### 1.5.2 ST-LINK/V2

ST사에서 보급한 STM32F4-DISCOVERY에는 ST-LINK/V2가 내장되어 있다. 이 절에서는 STM32F4-DISCOVERY를 중심으로 기술을 한다. ST-LINK/V2를 사용하여 debugging and programming 을 하기 위해서 보드의 USB 커넥터 CN1을 통해 PC에 STM32F4-DISCOVERY 보드를 연결한다.

**STM32F4-DISCOVERY**보드에서 ST-LINK/V2는 STM32 devices(*STM8 and STM32*)에 대한 **SWD**만 지원한다. 시중에 판매되는 ST-LINK/V2는JTAG 지원한다. 또한 ST외 타사의 디바이스는 지원을 하지 않는다

ST 홈페이지에서 자료 받기

Related Tools and Software

| Part Number | Description |
|---|---|
| STSW-LINK003 | ST-LINK/V2 USB driver for Windows 7, Vista and XP |
| STSW-LINK004 | STM32 ST-LINK utility |

STSW-LINK003
http://www.st.com/web/en/catalog/tools/PF258167

STSW-LINK004
http://www.st.com/web/en/catalog/tools/PF258168

#### 1.5.2.1. STM32F4DISCOVERY 보드 ST-LINK/V2 설정

| Jumper state | Description |
|---|---|
| 둘 다 CN3 점퍼 핀 ON | ST-LINK/V2( board programming) |
| 둘 다 CN3 점퍼 핀 OFF | ST-LINK/V2 ( external CN2) |

STM32F4DISCOVERY 보드 SWD 핀 번호

| Pin | CN2 | Designation |
|---|---|---|
| 1 | VDD_TARGET | VDD from application |
| 2 | SWCLK | SWD clock (JTCK) |
| 3 | GND | Ground |
| 4 | SWDIO | SWD data input/output (JTMS) |
| 5 | NRST | RESET of target MCU |
| 6 | SWO | Reserved (JTDO) |

아래는 일반적인 JTAG 기능과 ST-LINK/V2 와 비교한 것이다.

| Pin no. | ST-LINK/V2 connector (CN3) | ST-LINK/V2 function | Target connection (SWD) | Target connection (JTAG) |
|---|---|---|---|---|
| 1 | VAPP | Target VCC | MCU VDD(1) | MCU VDD(1) |
| 2 | | | | |
| 3 | TRST | JTAG TRST | GND(2) | JNTRST |
| 4 | GND | GND | GND(3) | GND(3) |
| 5 | TDI | JTAG TDO | GND(2) | JTDI |
| 6 | GND | GND | GND(3) | GND(3) |
| 7 | TMS_SWDIO | JTAG TMS, SW IO | SWDIO | JTMS |
| 8 | GND | GND | GND(3) | GND(3) |
| 9 | TCK_SWCLK | JTAG TCK, SW CLK | SWCLK | JTCK |
| 10 | GND | GND | GND(3) | GND(3) |
| 11 | NC | NC | NC | NC |
| 12 | GND | GND | GND(3) | GND(3) |
| 13 | TDO_SWO | JTAG TDI, SWO | TRACESWO(4) | JTDO |
| 14 | GND | GND | GND(3) | GND(3) |
| 15 | NRST | NRST | NRST | NRST |
| 16 | GND | GND | GND(3) | GND(3) |
| 17 | NC | NC | NC | NC |
| 18 | GND | GND | GND(3) | GND(3) |
| 19 | VDD | VDD (3.3V) | NC | NC |
| 20 | GND | GND | GND(3) | GND(3) |

1. The power supply from the application board is connected to the ST-LINK/V2 debugging and programming
board to ensure signal compatibility between both boards.
2. Connect to GND for noise reduction on the ribbon
3. At least one of this pin must be connected to the ground for correct behavior (connecting all of them is recommended)
4. Optional: for Serial Wire Viewer (SWV) trace

1.5.2.2 ST-LINK/V2 사용법(컴파일 내)

컴파일러 Tool 에서 다운로드 할 경우 아래 버전 이상에서만 지원이 가능하다. ST-LINK/V2 와 STM32F4 디바이스가 최근에 나왔기 때문이다.

| Third party | Toolchain | Version |
|---|---|---|
| Atollic | TrueSTUDIO | 2.1 |
| IAR | EWARM | 6.20 |
| Keil | MDK-ARM | 4.20 |
| TASKING | VX-toolset for ARM Cortex-M | 4.0.1 |

ST-LINK/V2 드라이버를 설치가 되어 있어야 한다. keil 프로그램을 실행하고 Alt+F7 를 눌러 아래 화면의 Debug 로 들어간다.

- STM32F4-DISCOVERY 보드를 사용할 경우 Flash Download 화면에서 Add를 클릭하여STM32F4x Flash를 선택한다.



그러면 아래 그림처럼 Programming Algorithm 에 디바이스가 선택된 것이 보인다.

STM32F4-DISCOVERY보드의 ST-LINK 로 다운로드 하기

| Jumper state | Description |
|---|---|
| 둘 다 CN3 점퍼 핀 OFF | ST-LINK/V2 ( external CN2) |

STM32F4DISCOVERY 보드 SWD 핀 번호와 RE, RC 보드 연결

| Pin | CN2 | RE, RC 보드 |
|---|---|---|
| 1 | VDD_TARGET | 1-VCC |
| 2 | SWCLK | 5-TCK |
| 3 | GND | 6-GND |
| 4 | SWDIO | 4-TMS |
| 5 | NRST | 8-/RST |
| 6 | SWO | 7-TDO |

- Stm32F103RE 보드 다운로드 하기 전에 S2 스위치를 EX 에 위치에 놓고 한다. 다운로드 후 리셋한다.

- Stm32F103RC 보드 다운로드 하기 전에 S2 스위치를 EX 에 위치에 놓고 한다. 다운로드 후 리셋한다.

Debug 화면에서 Port에서 SW를 선택하면 SW Device 화면에 아래 그림처럼 Device가 인식된 것 알 수 있다.

지금까지 ST-Link 설정이 완료되었다.

아래 LOAD 를 클릭하면 내부 프로그램을 지우고 프로그램을 다운로드 한다.

### 1.5.2.2 ST-LINK/V2 사용법(STM32 ST-LINK Utility )

keil 컴파일러 에서 다운로드가 아닌 다운로드 전용 프로그램으로 다운로드 방법을 알아본다.
오직 프로그램을 지우고, 다운로드하는 전용 프로그램이다.
http://www.st.com/internet/evalboard/product/251168.jsp 에 들어가서
아래 프로그램을 설치한다.

1) STM32 ST-LINK Utility
2) ST-LINK/V2 USB driver for Windows 7, Vista and XP

st-link_v2_usbdriver.exe 드라이버를 설치해야 장치관리자에 그림처럼 등록이 된다.

[그림 1.5.13] 장치관리자 등록

그리고 바탕화면에 프로그램을 실행한다.

[그림 1.5.14] 바탕화면 실행 파일

STM32 ST-LINK utility 2.3.0 을 설치하였는데 2.2.5 이하 버전은 STM32F4-Discovery 의
CPU 지원이 안되기 때문에 그 이상 버전을 설치한다. ST 홈페이지에는 2.3.0 을 제공한다.

아래 그림은 설치 후 Help => About 을 클릭하면 나온다.

[그림 1.5.15] utility 2.3.0 버전

아래 메뉴에서 Target => Settings 들어가 클릭하거나, 아래 그림처럼 Settings 그림을 클릭한다.

[그림 1.5.16] 실행화면

아래그림에서 SWD 체크 하고 OK 를 클릭한다.

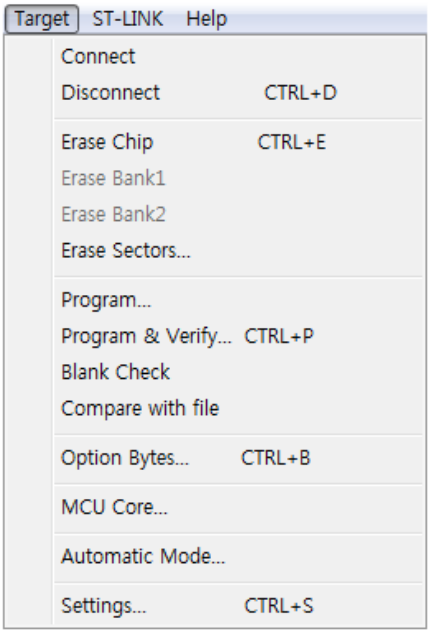



[그림 1.5.17] Settings 화면

아래[그림 1.5.18]처럼 나오면 정상이다. 아래 그림 대신 error 창에 " No Target connected" 문구가 나오면 연결이 안된 상태이다. 이런 경우는 장치관리자에 프로그램이 제대로 설치되지 않을 때는 프로그램을 다시 설치한다. 또는 케이블이 불량이거나, 하드웨어 설정불량 등을 알 수 있다.

[그림 1.5.18] 장치인식

메뉴에서 Target를 클릭한다.




[그림 1.5.19 ] 다운로드 주 화면

Program.. 을 클릭하여 다운로드를 한다.

[그림 1.5.20] 다운로드 화면

ST-LINK/V2 사용하여 다운로드

아래는 ST-LINK/V2 와 RE, RC 보드 연결 핀이다..

| Pin no. | ST-LINK/V2 connector (CN3) | ST-LINK/V2 function | RE, RC 보드 (SWD) | RE, RC 보드 (JTAG) |
|---|---|---|---|---|
| 1 | VAPP | Target VCC | 1-VCC | 1-VCC |
| 2 | | | | |
| 3 | TRST | JTAG TRST | GND | 2-TRS |
| 4 | GND | GND | GND | GND |
| 5 | TDI | JTAG TDO | GND | 3-TDI |
| 6 | GND | GND | GND | GND |
| 7 | TMS_SWDIO | JTAG TMS, SW IO | 4-TMS | 4-TMS |
| 8 | GND | GND | GND | GND |
| 9 | TCK_SWCLK | JTAG TCK, SW CLK | 5-TCK | 5-TCK |
| 10 | GND | GND | GND | GND |
| 11 | NC | NC | NC | NC |
| 12 | GND | GND | GND | GND |
| 13 | TDO_SWO | JTAG TDI, SWO | 7-TDO | 7-TDO |
| 14 | GND | GND | GND | GND |
| 15 | NRST | NRST | 8-/RST | 8-/RST |
| 16 | GND | GND | GND | GND |
| 17 | NC | NC | NC | NC |
| 18 | GND | GND | GND | GND |
| 19 | VDD | VDD (3.3V) | NC | NC |
| 20 | GND | GND | GND | GND |

### 1.5.3 H-JTAG 패럿포트



[그림 1.5.20] 패럿포트 실물사진

H-JTAG (패럿포트용) 제품은 http://www.hjtag.com 에서 지원하는 하드웨어와 응용프로그램을 무상으로 지원한다. 요즘 패럿포트가 없기 때문에 권장하지 않는다.

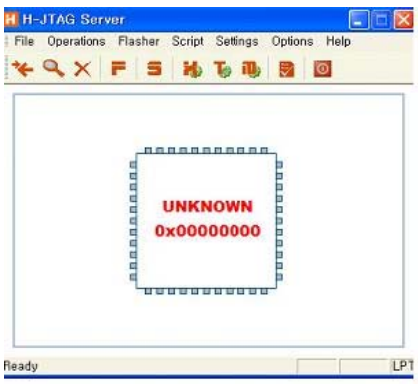
H-JTAG 의 사용법은 MULTI-ICE 제품과 유사하며 SDT2.51, ADS1.2, REALVIEW(KEIL, RDS) and IAR 등과 같은 대부분의 인기 있는 개발 환경에서 ARM7/ARM9/Cortex-M3 코어의 디버깅 및 플래시 프로그램을 할 수 있다.

**H-jtag 사용하기**

http://www.hjtag.com/download.html 사이트에 들어가 H-JTAG 최신버전을 다운받아 설치한다. 무료 공개 소프트웨어이다.
ARM 보드를 연결하고 전원을 인가하고 아래 그림처럼 H-JTAG 를 클릭하여 프로그램을 실행한다.


[그림 1.5.21] H-JTAG 실행 파일

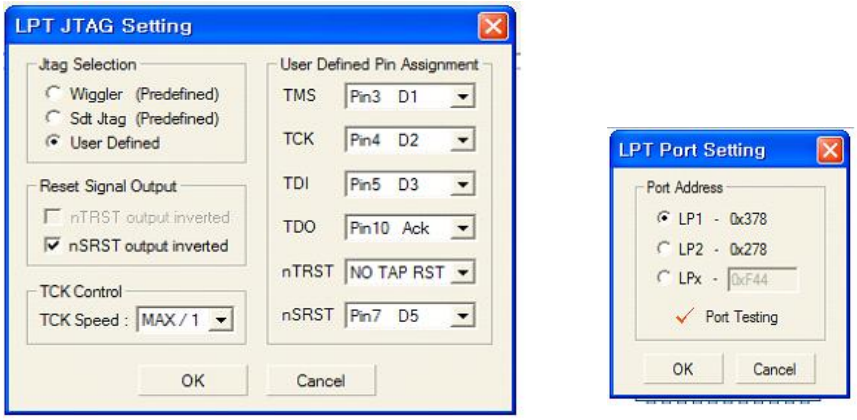
화면이 나오면 환경 설정이 안되어 있기 때문이다.




[그림 1.5.22] H-JTAG 환경 설정

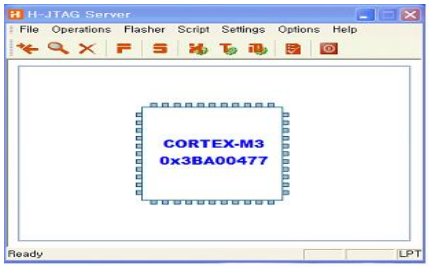
메뉴의 Settings 에서


[그림 1.5.23] H-JTAG 포트 설정



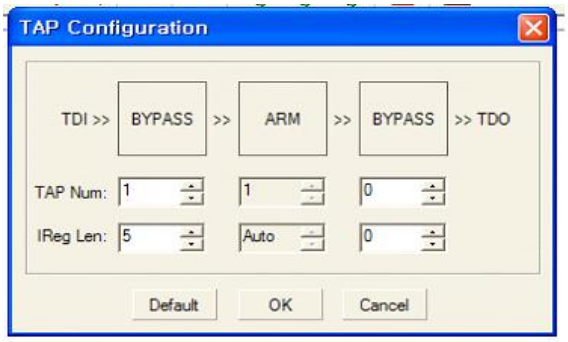

[그림 1.5.24] H-JTAG 핀 및 포트 설정

여기까지 설정하면 [그림 1.5.18]처럼 H-JTAG 디바이스 인식해 ARM 코아 를 디텍터 한다.




[그림 1.5.25] H-JTAG 디바이스 인식

STM32F103F8 인경우 메뉴의 Settings 에서 TAP Configuration 에서 아래 그림처럼 설정한다.

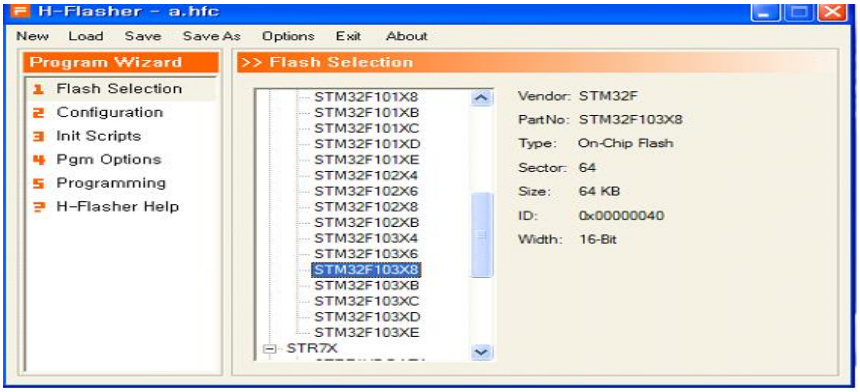



[그림 1.5.26] TAP Configuration 설정

그리고 메뉴의 Flasher 에서 Start H-Flasher 에서

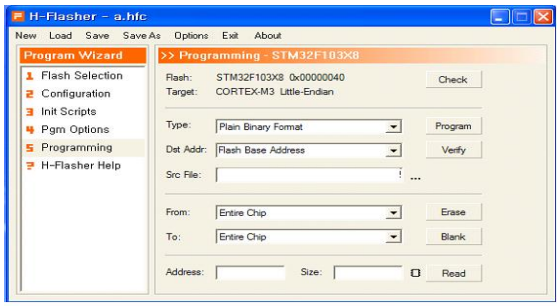
[그림 1.5.27] H-Flasher 실행파일

디바이스에 맞게 설정한다. 여기서는 STM32F103F8 일 때이다. Program Wizard 에서 1. Flash Selection 에서 STM32F103F8 사용하므로 STM32F103x8 을 선택한다.



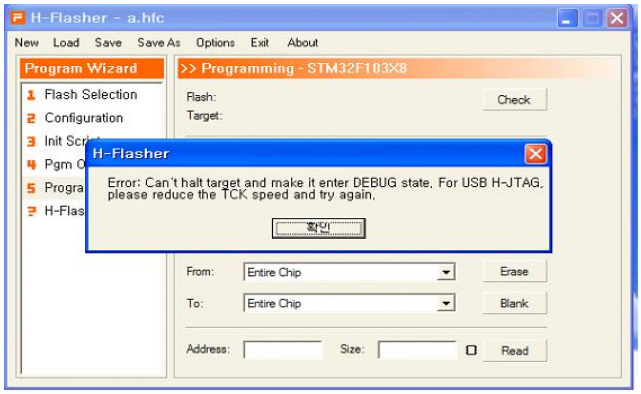

[그림 1.5.28] 디바이스 설정

5. Programming 에서 체크를 하면 아래그림 처럼 Flash, Target 에 디스플레이 된다.




[그림 1.5.29] Programming

제대로 설정되지 않으면 아래 그림처럼 된다.




[그림 1.5.30] 설정이 안된 경우

마지막으로 Type, Dst Addr 을 설정하고 Src File: 에서 다운로드할 파일을 설정해 다운로드하면 된다.

## 1.6 프로그램 시뮬레이션

ARM 컴파일에는 여러 종류가 있으나, Keil 사의 RealView 에는 하드웨어가 없어도 미리 결과를 알수 있는 시뮬레이션이 있어서 편리하다. 여기서는 컴파일 에러 없이 된 후 가능하다. [그림1.6.1] 처럼 왼쪽 Project 아래 Test 를 클릭하고 Option for Target 'Test'… 를 선택한다.

[그림1.6.1] Option 설정

[그림1.6.2] 아래 그림처럼 Use Simulator 가 체크되어 있어야 시뮬레이션을 할 수 있다.

[그림1.6.2] Simulator 체크

그리고 아래 [그림1.6.3] 처럼 메뉴 바의 Debug 아래 툴을 선택하거나

[그림1.6.3] Simulator Debug 선택 1

[그림1.6.4] 그림 도구 바의  을 선택한다.

[그림1.6.4] Simulator Debug 선택 2

그림 도구 바의  Logic Analyzer 을 선택하고  화면을 클릭하면 아래그림[그림1.6.5] 처럼 화면이 뜬다. 여기서  클릭하여 출력 포트 PORTB.8 을 쓰고 창을 닫는다.

[그림1.6.5] 포트 입력

시뮬레이션 결과를 보기 위해  클릭 또는 Run(F5)을 하면 [그림1.6.6]와 같이 파형 결과를 볼 수 있다. 파형을 전체 볼 때는  을 클릭하고, 리셋을 할 때  클릭하고, 중지할때  클릭하면 된다.

[그림1.6.6] 파형 결과

그리고 또 다른 결과 보는 법은 아래 [그림1.6.7]의 포트 설정설정으로 들어간다.

[그림1.6.7] 포트 설정

아래 [그림1.6.8]처럼 포트 PORTB.8 이 체크 상태를 반복하는 것을 알 수 있다.

[그림1.6.8] 체크박스로 본 결과

실습편에 있는 2_GPIO_LED 를 통해 포트 2개를 시뮬레이션 해 본다. 아래 그림 전 까지는 앞 전에 했던 것과 동일하다. 아래 [그림1.6.7]에서 PORTB.1과 PORTB.8 을 쓰고 창을 닫고 실행한다.

[그림1.6.7] 한 개 이상 포트 설정

아래 [그림1.6.8]처럼 포트 PORTB.1과 PORTB.8 이 반전 상태를 알 수 있다.

[그림1.6.8] 포트 2개 파형 결과

이번에는 USART 통신 프로그램을 시리얼 포트로 문자 전송하는 것을 시뮬레이션 하는 과정이다.
예제 프로그램 4_UASRT, 4_RCC_UASRT, 4_SIZEOF_USART 등을 실행 해 보자

[그림1.6.9] UART 시뮬레이션 설정

[그림1.6.9] 상단 오른쪽은 1. 디버그 시작과 끝을 할 때 사용하고, 2. UART 선택한다. 3. 시뮬레이션 결과를 실행 하기 위해 클릭하고, 4. 프로그램을 작성한 것을 출력해 볼 수 있다. 5. 리셋을 클릭해 다시 실행된 결과를 볼 수 있다. 지금까지 테스트 해 본 결과 하드 디버깅도 편하고 장비에 다운로드 안하고도 미리 알 수 있어 개발에 편리하다.

# 제 2 장 하드웨어 구성

Main Board 보드와 Sub Board로 나누어져 있다. 별도로 따로 사용해도 된다.

## 2.1 Main Board

[그림2.1.1] 메인보드 전체 회로도이다.

[그림2.1.1] 메인보드 회로도

[그림2.1.2] 메인보드 에서 왼쪽은 USB-B/Female 라이트 앵글(DS 1099-W) 타입이고, 오른쪽 Mini-USB SMT(5pin)-AB 타입이다. 둘 중하나 장착하게 되어 있다. Cortex-M3 의 STM32F103 계열의 64핀 LQFP 타입을 중심으로 구성되어있다. 보드 사이즈는 78x58 mm 로 되어있다.

[그림2.1.2] 메인보드

**CPU(STM32F103RCT6, RET6) 보드**

**1) CPU 부**

ST사 Cortex-M3의 STM32F103 계열을 사용하였다. PCB 에 핀 수 64pin이 호환 가능한. LQFP 타입이다. 위 PCB 보드에 가능한 STM32F103 계열 STM32F103R8T6, STM32F103RCT6, STM32F103RET6이다. 여기서는 8MHz 크리스털을 사용하였다.

STM32 디바이스 모델로 사양을 알아보자.

STM32 => ARM-based 32-bit microcontroller

F => General Purpose

103 => Sub Family Number

R,V,Z => **R = 64pin**, V = 100pin, Z = 144 pin;

**C**, D, **E** => C = **256Kbyte**, D = 384 Kbyte, E = **512 Kbyte**

H,T => H = BGA, **T = LQFP**

6,7 => 온도 범위 **6 = -40~85 ℃**, 7= -40~105 ℃

STM32F103xx family

| Pinout | Low-density devices | | Medium-density devices | | High-density devices | | |
|---|---|---|---|---|---|---|---|
| | 16 KB Flash | 32 KB Flash(1) | 64 KB Flash | 128 KB Flash | 256 KB Flash | 384 KB Flash | 512 KB Flash |
| | 6 KB RAM | 10 KB RAM | 20 KB RAM | 20 KB RAM | 48 or 64 KB(2) RAM | 64 KB RAM | 64 KB RAM |
| 144 | 2 × USARTs<br>2 × 16-bit timers<br>1 × SPI, 1 × I²C, USB, CAN, 1 × PWM timer<br>2 × ADCs | | 3 × USARTs<br>3 × 16-bit timers<br>2 × SPIs, 2 × I²Cs, USB, CAN, 1 × PWM timer<br>2 × ADCs | | 5 × USARTs<br>4 × 16-bit timers, 2 × basic timers<br>3 × SPIs, 2 × I²Ss, 2 × I2Cs<br>USB, CAN, 2 × PWM timers<br>3 × ADCs, 2 × DACs, 1 × SDIO<br>FSMC (100- and 144-pin packages(3)) | | |
| 100 | | | | | | | |
| 64 | | | | | | | |
| 48 | | | | | STM32 F103 RCT6 | | STM32 F103 RET6 |
| 36 | | | | | | | |

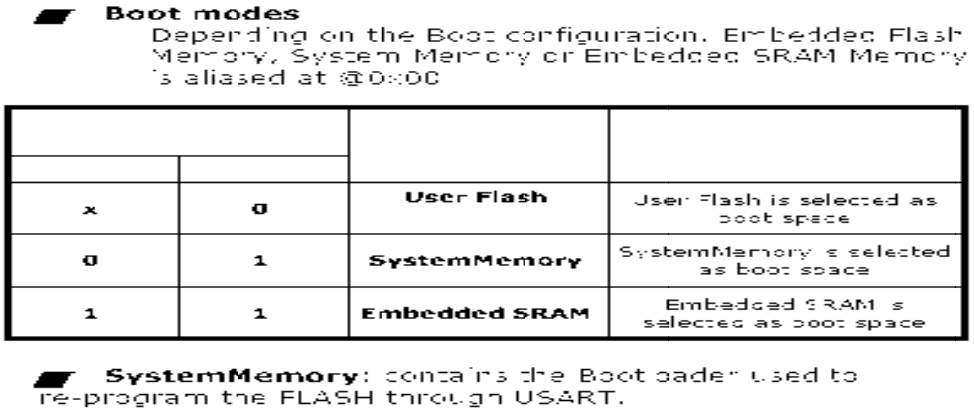
1. For orderable part numbers that do not show the A internal code after the temperature range code (6 or 7), the reference datasheet for electrical characteristics is that of the STM32F103x8/B medium-density devices.
2. 64 KB RAM for 256 KB Flash are available on devices delivered in CSP packages only.
3. Ports F and G are not available in devices delivered in 100-pin packages.

[그림2.1.2] STM32F103 계열

- Core: ARM 32-bit Cortex™-M3 CPU
  - **72 MHz maximum frequency**, 1.25 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state memory access
  - Single-cycle multiplication and hardware division
- Memories
  - **STM32F103RCT6 (256 Kbytes), STM32F103RCT6(512 Kbytes) - Flash memory**
  - **64 Kbytes of SRAM**
- Clock, reset and supply management
  - **2.0 ~ 3.6 V application supply and I/Os**
  - POR, PDR, and programmable voltage detector (PVD)
  - **4-to-16 MHz crystal 또는 oscillator**
  - Internal 8 MHz factory-trimmed RC
  - Internal 40 kHz RC with calibration
  - 32 kHz oscillator for RTC with calibration
- Low power
  - Sleep, Stop and Standby modes
  - VBAT supply for RTC and backup registers
- 3 × 12-bit, 1 µs A/D converters (up to 21 channels)
  - Conversion range: 0 to 3.6 V
  - Triple-sample and hold capability

  - Temperature sensor
- 2 × 12-bit D/A converters
- DMA: 12-channel DMA controller
  - Supported peripherals: timers, ADCs, DAC, SDIO, I2Ss, SPIs, I2Cs and USARTs
- Debug mode
  - **Serial wire debug (SWD) & JTAG interfaces**
  - Cortex-M3 Embedded Trace Macrocell™
- Up to 112 fast I/O ports
  - 51/80/112 I/Os, all mappable on 16 external interrupt vectors and almost all **5 V-tolerant**
- Up to 11 timers
  - Up to four 16-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
  - 2 × 16-bit motor control PWM timers with dead-time generation and emergency stop
  - 2 × watchdog timers (Independent and Window)
  - SysTick timer: a 24-bit downcounter
  - 2 × 16-bit basic timers to drive the DAC
- Up to 13 communication interfaces
  - Up to 2 × I2C interfaces (SMBus/PMBus)
  - Up to 5 USARTs (ISO 7816 interface, LIN, IrDA capability, modem control)
  - Up to 3 SPIs (18 Mbit/s), 2 with I2S interface multiplexed
  - CAN interface (2.0B Active)
  - **USB 2.0 full speed interface**
  - **SDIO interface**
- CRC calculation unit, 96-bit unique ID

**2) Boot Mode 부**



**Boot modes**

Depending on the Boot configuration, Embedded Flash Memory, System Memory or Embedded SRAM Memory is aliased at @0x00

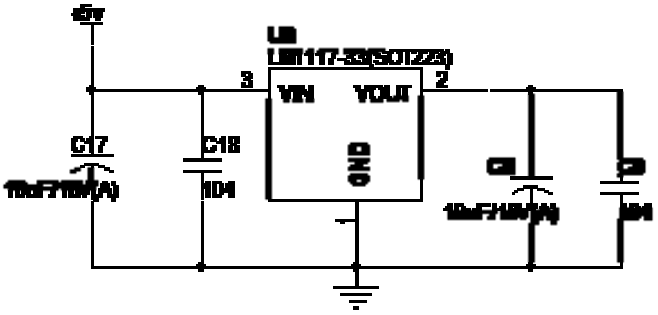
| | | | |
|---|---|---|---|
| x | 0 | **User Flash** | User Flash is selected as boot space |
| 0 | 1 | **SystemMemory** | SystemMemory is selected as boot space |
| 1 | 1 | **Embedded SRAM** | Embedded SRAM is selected as boot space |

**SystemMemory**: contains the Boot loader used to re-program the FLASH through USART.

[그림2.1.3] BOOT 모드

[그림2.1.3]은 보드 S2 에 EX 는 User Flash 로 다운로드 후 실행할 때 이고, DN은 System Memory로 CPU 내장 ROM으로 부트의 부트로 UART 를 통해 프로그램 다운로드 할 때 여기에 위치하고 다운로드를 한다. 아래 그림은 BOOT0으로 스위치 S2 를 왼쪽에 위치하면 다운로드 한 프로그램을 실행하고 오른쪽으로 위치하면 프로그램을 다운로드 한다. PB2 에 연결되어 있는 BOOT1은 10K 저항이 GND에 연결되어 있다.

3) 전원부




[그림2.1.4] 전원 회로

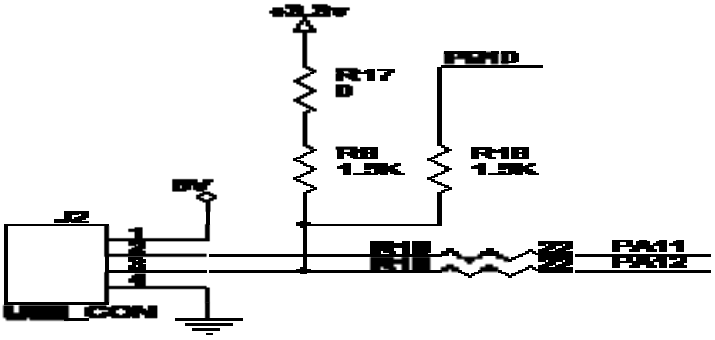
J9 커넥터의 보드 입력전압(아답터)은 일반적으로 5V 전원(**500mA이상**)을 사용한다. CPU전원이 3.3V 이므로 LDO 레귤레이터 3.3V인 LM1117 800mA사용하였다. 10uF 용도는 전원주변에 전원이 흔들리는 현상, 즉 리플을 제거 하기 위한 것이고, 104(0.1uF)는 디커플링 커패시터(Decoupling Capacirtor) 으로 여러 개의 스위칭 소자(논리소자)가 동시에 스위칭을 하면 순간적으로 많은 전류가 흘러 전원 전압이 떨어지는 현상이 발생합니다. 이 것을 방지하기 위해 각 IC의 전원과 그라운드에 커패시터를 달아주는 노이즈 제거용으로 많이 사용한다. 보드 입력전압은 일반적으로 5V 전원(**500mA이상**)을 사용한다. 단, USB 전원을 사용시 USB 용량이 최대 **500mA** 까지 사용할 수 있다.

아래 4가지 방법 중 한가지를 선택한다.
①별도의 5V 아답터 전원(1A이상)을 사용하여 J9에 연결한다.
**서브보드의 J12의 5V헤더핀을 제거한다. 그렇지 않으면 USB 전원과 충돌한다.**
②PC 의 USB 를 J2 에 연결하여 USB 전원 사용한다.
③다운로드용 USB232 사용하여 J2에 연결하여 전원을 사용하고 J6에 연결하여 또는 통신을 한다. (시그널테크에서 있는 USB232 제품 )
④TFT서브보드에는 USB232가 있어 전원과 통신을 할 수 있다.


[그림2.1.5] LED 회로

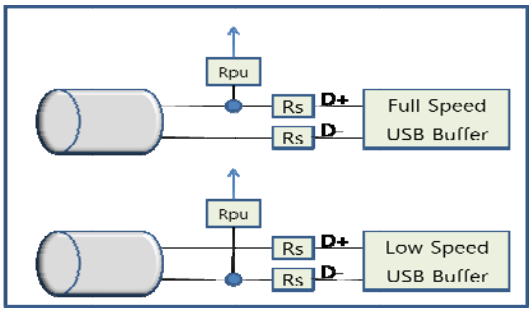
[그림2.1.5]은 D1 의 LED 를 통해 전원 인가 되어있다. 여기에 사용한 저항 값을 계산해 보자. LED 일반적으로 전류는 9mA 이고, LED에 거는 전압, 정격전압 2V 이다.
오옴의 법칙에 **R= V / I**
5V 일 때 5V - 2V / 9 mA =333.3333 일반적으로 330Ω 저항을 사용한다. 저항은 계산값의 근사치 사용. 3.3V 일 때 3.3V - 2V / 9 mA =133.3333 일반적으로 120Ω 저항을 사용한다. 실제 120 Ω 연결하면 너무 밝다. 여기서는 3.3V 이므로 120Ω 사용해야 하나 필요한 전류만 주기 위해 470 Ω 저항을 사용했다.

4) USB 부




[그림 2.1.6] USB 회로도

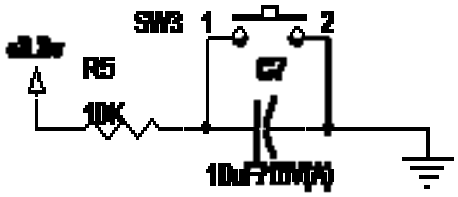
J2에 사용할 수 있는 USB 커넥터는 Mini-USB SMT(5pin)-AB 타입 또는 USB-B/Female 라이트 앵글(DS 1099-W) 타입 2 종류 중 한가지로 되어 있다. 23 페이지에 그림을 보면 알 수 있다. USB 포트로부터 전원을 공급받아 사용 할 수 도 있고, 마우스, 조이스틱, 저장 매체 등 프로그램에 따라 각종 기능을 사용한다. PA11(D-), PA12(D+) 두 핀과 제어 핀 PB10을 사용했다. (PA12에 R8에 1.5K 풀 업을 하면 프로그램에 상관없이 풀 스피드로 잡혀있고, R8을 제거하고 R18에 PA12와 PB10에 1.5K를 연결하고, PB10을 HIGH를 주어 인식하는데 사용, hw_config.c 에서 void USB_Cable_Config (FunctionalState NewState) 연결 함수 )
USB Full Speed Device 동작은 PA12(D+)1.5K 풀 업을 하고, USB Low Speed Device 동작은 PA11(D-)1.5K 풀 업을 한다. 아래 [그림 2.1.7]을 참고하면 이해하기 쉬울 것 같다. PB10을 사용하지 바로 풀 업을 사용해도 된다.




[그림 2.1.7] USB Speed Device 동작 상태

5) Reset 부

장비를 초기화 할 때 리셋 버튼을 누른다. 아래 [그림 2.1.8] Reset 회로를 보면 평상시에는 High 에서 low 로 떨어질 때이다.

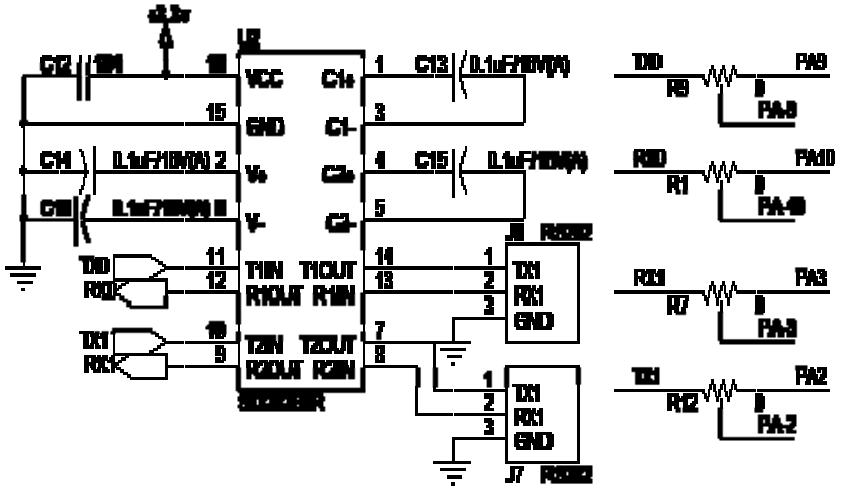



[그림 2.1.8] Reset 회로

**6) RS232 부**

USART 통신 프로그램으로 PC 와 통신을 한다. 메인보드의 S2 스위치를 DN 으로 놓으면 프로그램 다운로드 할 때 쓰이고, 평상시는 EX(실행)으로 위치해 놓고 통신 포트로 사용한다.

(1) PC의 시리얼포트 연결 방법

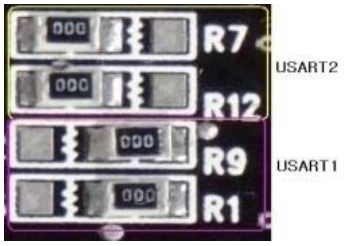



[그림 2.1.9] USART 통신 회로도

통신포트단자로 USART1, USART2를 사용할 수 있다. Main 보드 제품 출하 시는 USART1 만 가능 하다. USART1는 PA9(TX0), PA10(RX0) 에 연결 되어 있으며, USART2는 PA2(TX1), PA3(RX1) 에 연결 되어 있다., 통신포트로 사용시는 아래그림 처럼 저항을 **0 Ω** 을 오른쪽으로 위치하고, 커넥터 J7을 사용하면 된다.

| 위치 | 왼쪽 | 오른쪽 |
|---|---|---|
| **R1, R9** | **I/O** | **USART1** |
| R7,R12 | I/O | USART2 |

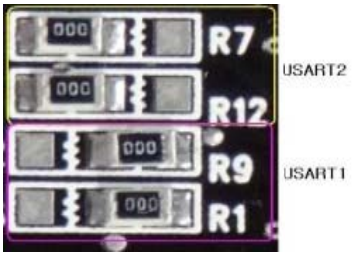
J6 의 TX0, RX0, GND 를 PC 의 시리얼포트의 D-SUB 9핀 커넥터 2번, 3번, 5번 순서대로 연결한다.




[그림 2.1.10] 시리얼 포트로 연결는 방법

(2) PC의 USB 포트 연결 방법

USB 와 RS-232신호를 변환을 장치를 이용하여 사용한다. 메인보드에서 저항 R1, R9 는 오른쪽에 위치한다.




[그림 2.1.11] USB 포트를 연결하는 방법

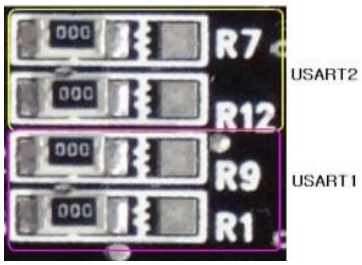
ARM 에서 TTL 신호를 RS232 로 변환 해 주는 IC Device 로 MAX232 는 전원이 5V 일 때, MAX3232 는 3.3V 일 때 사용한다. MAX 는 제조사 이므로 다른 동일 디바이스도 무관하다. 여기서는 사용하고 있는 Device 는 STM32F103 제조사인 ST 의 ST3232 를 사용하였다.

(3) TFT 서브보드 이용 방법

STM32F103 포트에서 바로 PLX2303 으로 연결되어 USB 통신을 한다. 메인보드와 서브보드 같이 사용시는 TFT**서브 보드를 이용해 USB 포트로 다운로드 할 경우 0 Ω 저항을 모두 왼쪽에 배치 된다.** 저항설정은 아래그림처럼 왼쪽으로 0오옴이 배치 된다. ARM 보드와 TFT 서브보드 연결 되어 있는데 이중에서 서브보드 J12에 USB

전원이 메인보드에 연결되어 전원을 공급하고, 서브보드 J14를 통해 통신 및 다운로드 할 수 있게 되어 있다.






[그림 2.1.12] TFT 서브보드를 이용하는 방법

### 7) JTAG

JTAG(Joint Test Action Group)는 시스템 개발 시에 사용하는 디버깅 장비이다. JTAG의 작동 방식은 칩 내부에 Boundary Cell을 만들어 이것이 외부의 핀과 일대 일로 연결되어, 프로세서가 할 수 있는 동작을 중간의 Cell을 통해 인위적으로 수행할 수 있도록 하는 것이다. 이런 방식으로 JTAG은 다양한 하드웨어의 테스트나 연결 상태등을 체크할 수 있다.

JTAG 인터페이스는 다음과 같은 핀으로 칩 안에 구성된다.

1.TDI (데이터 입력) : Test하기 위한 데이터 신호. TMS에 의해 전이된 TAP state에 따라, TDI가 command/data 가 결정됨

2.TDO (데이터 출력) : Test한 결과를 외부에서 모니터링 하기 위한 pin, 이 역시 TAP state에 따라 address/data가 될 수 있음.

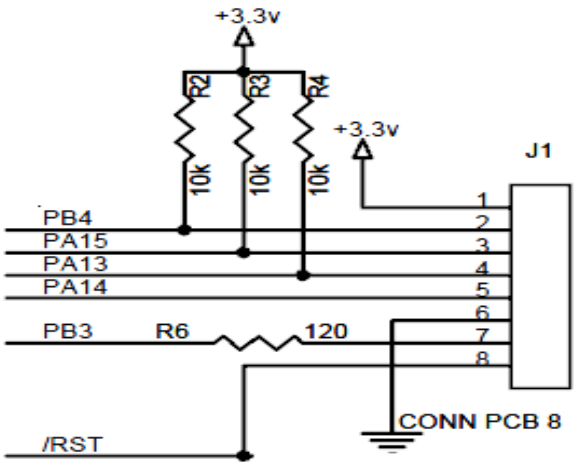
3.TCK (클럭) : Test clock

4.TMS (모드) : Test Mode 전환하기 위한 제어 신호

5.TRST (리셋)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| VCC | *TRST* | *TDI* | TMS | TCK | GND | TDO | /RST |
| VCC | *PB4* | *PA15* | PA13 | PA14 | GND | PB3 | /RST |

여기서 위의 2(TRST), 3(TDI)핀을 제외하고, 사용하면 SWD 으로 동작한다.
ST-Link 사용시 연결하면 개발 시 편리하다.




[그림 2.1.13] JTAG 연결 핀

아래는 일반적인 JTAG/SWD connectors 이다.

| **JTAG** | **SWD(최근)** | 10pts(ARM JTAG/SWD | 20pts (old ARM JTAG/SWD) | 24 pts(RLink3) |
|---|---|---|---|---|
| **GND** | **GND** | 3,5,9 | 4,6,8,10,12,14,16,18,20 | 3,4,10,17,19,21,22 |
| **VCC **** | **VCC **** | 1 | 1,2 | 1 |
| **TRST*** | **TRST*** | - | 3 * | 6* |
| **TDI** | **-** | 8 | 5 | 8 |
| **TMS** | **SWDIO** | 2 | 7 | 12 |
| **TCK** | **SWDCLK** | 4 | 9 | 13 |
| **RTCK** | **-** | - | 11 | 5 |
| **TDO** | **-** | 6 | 13 | 15 |
| **RST*** | **RST*** | 10* | 15 * | 11* |
| **DBGRQ** | **-** | - | (17) do not connect | 9 |
| **DBGACK** | **-** | - | 19 | 16 |
| **-** | **-** | - | - | 7, 14 |

RST and TRST are two different signals. They MUST NOT be connected to each other!
RST (also sometimes called NRST) is required for all devices.
TRST (also sometimes called nTRST, JTRST, nJTRST or JNTRST) is required for uPSD and PPC devices, and for most ARM devices.
You MUST connect these two signals from RLink to the target CPU if they are available.

**8) I/O 부**

POART A, POARTB, POARTC 를 사용 할 수 있게 커넥터로 구성되어 있다.

J3 커넥터 POARTA

| GND | PA1 | PA3 | PA5 | PA7 | PA9 | PA11 | PA13 | PA15 |
|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
| 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 |
| 3.3V | PA0 | PA2 | PA4 | PA6 | PA8 | PA10 | PA12 | PA14 |

J4 커넥터 POARTB

| GND | PB1 | PB3 | PB5 | PB7 | PB9 | PB11 | PB13 | PB15 |
|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
| 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 |
| 3.3V | PB0 | PB2 | PB4 | PB6 | PB8 | PB10 | PB12 | PB14 |

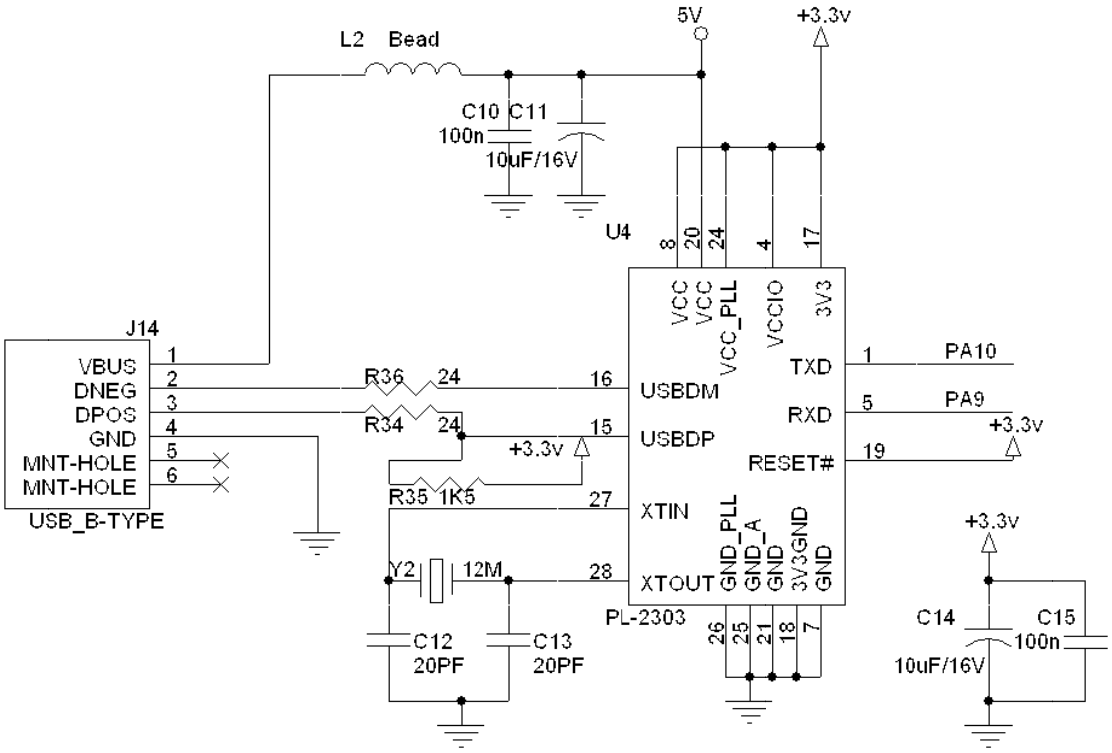
J5 커넥터 POARTC

| GND | PC1 | PC3 | PC5 | PC7 | PC9 | PC11 | PC13 | PC15 |
|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
| 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 |
| 3.3V | PC0 | PC2 | PC4 | PC6 | PC8 | PC10 | PC12 | PC14 |

## 2.2 Sub Board

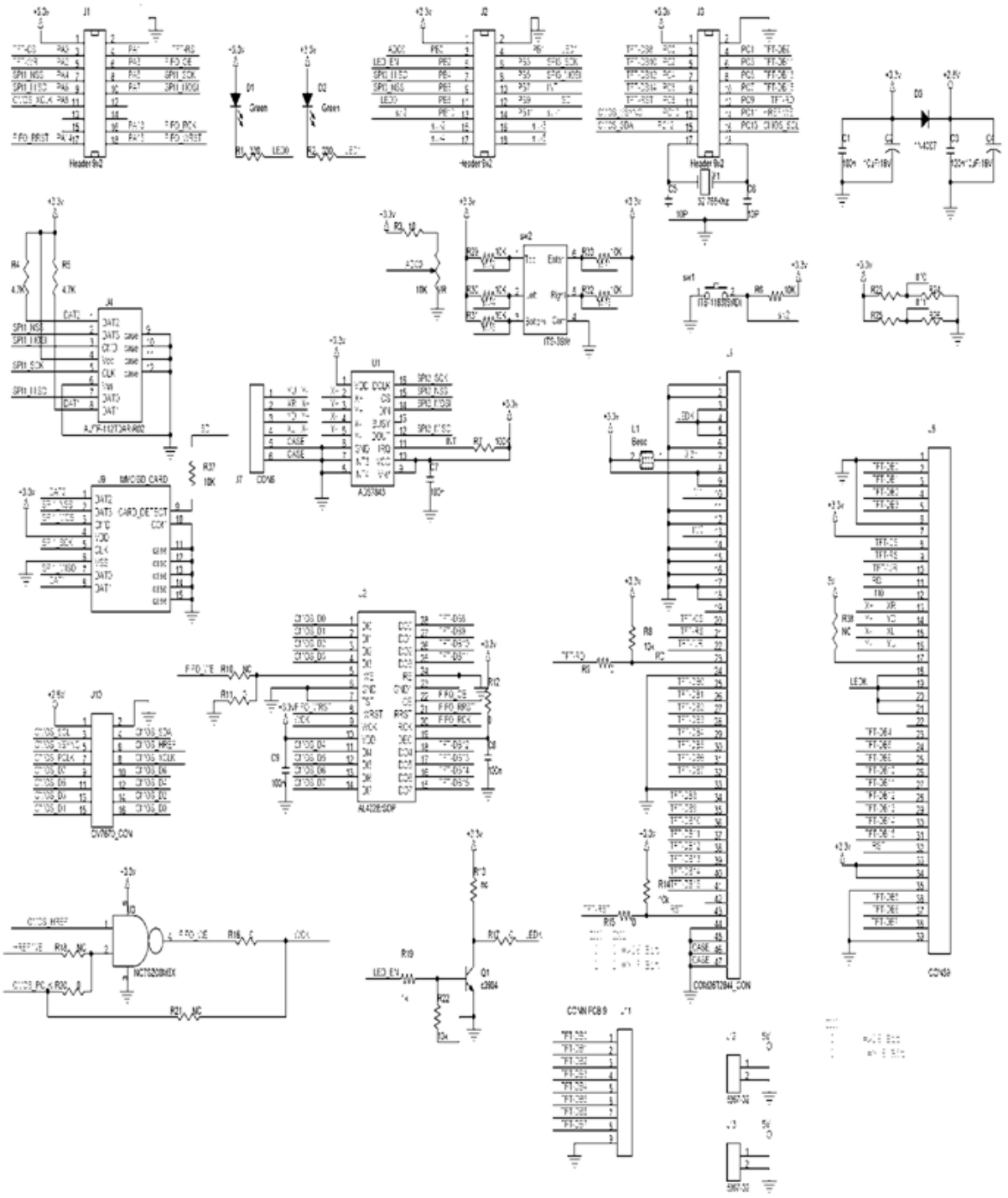
STM32F103 테스트보드로 LED, AD, SW, TFT LCD, RTC, SD Card, 카메라를 테스트 할 수 있게 구성되어 있다.

[그림 2.2.1] 서브 보드 실물



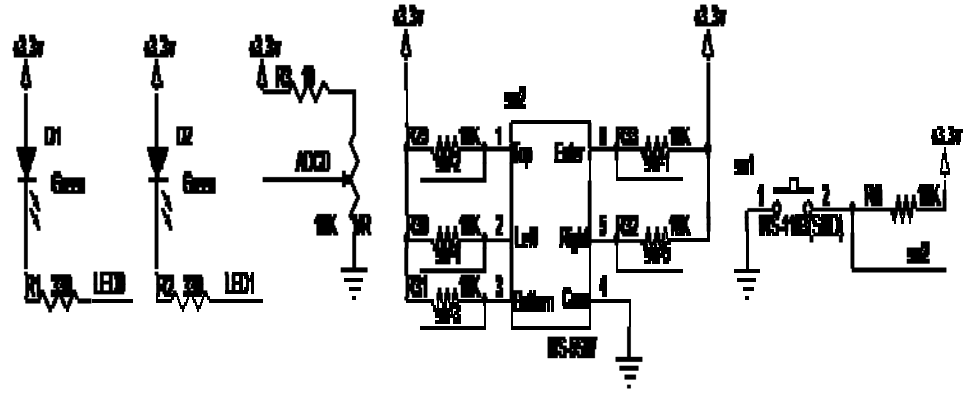

[그림 2.2.2] 서브 보드 회로도 (USB 회로도)

위그림은 PL-2302X 회로도이고, PL-2303TA 인 경우 칩만 교체해도 된다. Data sheet 대로 할려면 R36, R34 사용하지 않고 0오옴처리, R35는 제거한다.
PL-2303TA 핀 8,24번핀은 연결하지 않는다.



[그림 2.2.3] 서브보드 회로도

**1) 테스트 보드 (LED, ADC, SW)**



[그림 2.2.4] LED, ADC, SW 회로도

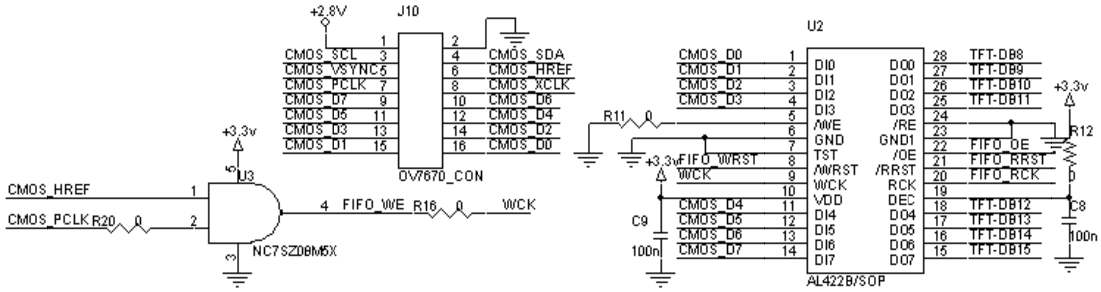
PB1 은 LED1에 PB8 은 LED0 에 연결되어 있고, PB1, PB8 이 0 (LOW) 일 때 LED ON 상태이다. PB0 은 ADC0 연결되어 있고, AD 변환을 테스트 할 수 있다.
[그림 2.2.5] 처럼 SW2(**ITS-1500S 모델**) 는 PB11=>SW-1, PB12=>SW-2, PB13=>SW-3, PB14=>SW-4, PB15=>SW-5 로 연결되어 있으며, 풀 업 상태이며, 스위치 5개가 연결되어 있다.
SW2( **ITS-1500S 모델** ) 일명 조이스틱 스위치라고도 한다. JoyStickMouse 이라는 예제 프로그램을 다운로드 하면, 조이스틱 역할을 한다. PC 모니터의 마우스에 위치한 커서가 이 조이스틱스위치를 통해 상하좌우 움직인다.



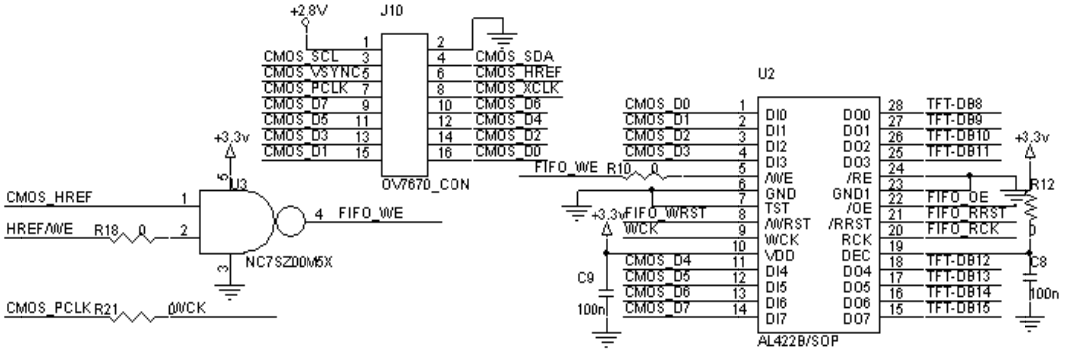
[그림 2.2.5] 조이스틱 스위치

SW1는 PB10 연결되어 있으며, 스위치 역할을 하거나, USB 의 연결을 끊었다, 붙였다 하는 USB 인식하는데 사용하기도 한다. 현재는 SW 로는 사용하지 않고 USB 컨트롤 핀으로 쓰고 있다.

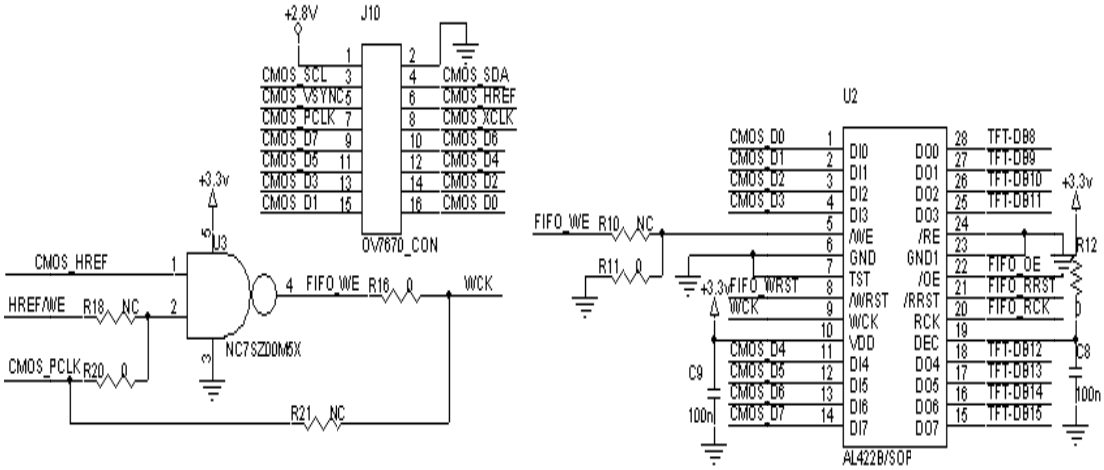
**2) OV7670 부**
OV7670 카메라 회로도 아래 [그림 2.2.6-1]과 [그림 2.2.6-2]가 있는데, 카메라 모듈 주변 회로도 1,2 두 가지 프로그램을 사용한다. 용도에 맞게 한가지를 선택한다. [그림 2.2.6-3] 카메라 모듈 주변 회로도 3 는 둘 중 한가지를 사용하도록 회로도, PCB 에 되어 있다.




[그림 2.2.6-1] 카메라 모듈 주변 회로도 1




[그림 2.2.6-2] 카메라 모듈 주변 회로도 2



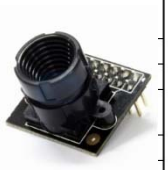

[그림 2.2.6-3] 카메라 모듈 주변 회로도 3

| PA8 | PA13 | PA14 | PA15 | PC10 | PC11 | PC12 | PC13 |
|---|---|---|---|---|---|---|---|
| XCLK | FIFO_RD | FIFO_RRST | FIFO_WRST | 7670_VSYNC | 7670_HREF | SCCB_SID | SCCB_SIC |

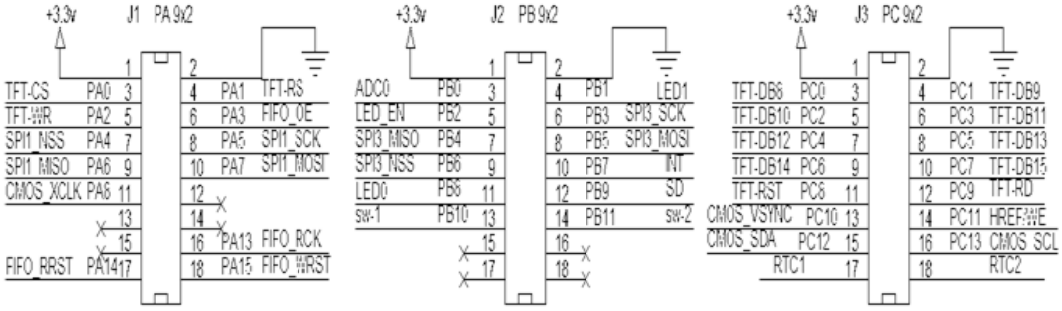
OV7670 카메라 연결될 CPU 핀 (데이터 핀은 AL422AL 에서 연결)

OV7670스펙

| | | |
|---|---|---|
| Active Array Size | | 640 x 480 |
| Power Supply | Digital Core | 1.8VDC ±10% |
| | Analog | 2.45V to 3.0V |
| | I/O | 1.7V to 3.0V[a] |
| Power Requirements | Active | 60 mW typical (15fps VGA YUV format) |
| | Standby | < 20 µA |
| Temperature Range | Operation | -30°C to 70°C |
| | Stable Image | 0°C to 50°C |
| Output Formats (8-bit) | | • YUV/YCbCr 4:2:2<br>• RGB565/555/444<br>• GRB 4:2:2<br>• Raw RGB Data |
| Lens Size | | 1/6" |
| Chief Ray Angle | | 25° |
| Maximum Image Transfer Rate | | 30 fps for VGA |
| Sensitivity | | 1.3 V/(Lux • sec) |
| S/N Ratio | | 46 dB |
| Dynamic Range | | 52 dB |
| Scan Mode | | Progressive |
| Electronics Exposure | | Up to 510:1 (for selected fps) |
| Pixel Size | | 3.6 µm x 3.6 µm |
| Dark Current | | 12 mV/s at 60°C |
| Well Capacity | | 17 K e |
| Image Area | | 2.36 mm x 1.76 mm |
| Package Dimensions | | 3785 µm x 4235 µm |



### 3) I/O 커넥터(PA, PB, PC)




- J1 커넥터

| PA0 | PA1 | PA2 | PA3 | PA4 | PA5 | PA6 | PA7 |
|---|---|---|---|---|---|---|---|
| LCD_CS | LCD_RS | LCD_WR | FIFO_CS | SD_CS | SPI_SCK | SPI_MISO | SPI_MOSI |

| PA8 | PA9 | PA10 | PA11 | PA12 | PA13 | PA14 | PA15 |
|---|---|---|---|---|---|---|---|
| XCLK | TX1 | RX1 | USB | USB | FIFO_RD | FIFO_RRST | FIFO_WRST |

- J2 커넥터

| PB0 | PB1 | PB2 | PB3 | PB4 | PB5 | PB6 | PB7 |
|---|---|---|---|---|---|---|---|
| ADC0 | LED1 | LED_EN | SPI_CLK | SPI_MISO | SPI_MOSI | TP_CS | TP_INT |

| PB8 | PB9 | PB10 | PB11 | PB12 | PB13 | PB14 | PB15 |
|---|---|---|---|---|---|---|---|
| LED0 | SD | SW1 | SW-1 | SW-2 | SW-3 | SW-4 | SW-5 |

- J3 커넥터

| PC0 | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 |
|---|---|---|---|---|---|---|---|
| LCD_D8 | LCD_D9 | LCD_D10 | LCD_D11 | LCD_D12 | LCD_D13 | LCD_D14 | LCD_D15 |

| PC8 | PC9 | PC10 | PC11 | PC12 | PC13 | PC14 | PC15 |
|---|---|---|---|---|---|---|---|
| LCD_RST | LCD_RD | 7670_VSYNC | 7670_HREF | SCCB_SID | SCCB_SIC | RTC | RTC |

**4) J11 커넥터**

| LCD_D0 | LCD_D1 | LCD_D2 | LCD_D3 | LCD_D4 | LCD_D5 | LCD_D6 | LCD_D7 |
|---|---|---|---|---|---|---|---|

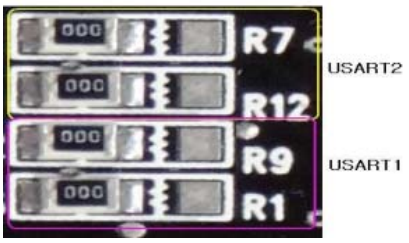
16비트로 사용할 경우 사용하는데, 여기서는 8비트로 사용하지 않았다.

**5) J12 커넥터**

J14 의 USB 전원과 연결되어 있으며 5V, GND 로 되어 있어 메인보드 전원을 공급하게 되어 있다. **J12의 5V헤더핀을 옆으로 눕히거나 제거한다. 그렇지 않으면 USB 전원과 충돌한다.**

**6) J14 커넥터**

USB 단자로 ARM 보드와 TFT 서브보드과 함께 연결되어 있다. 서브보드에 USB 전원이 메인보드에 연결되어 전원을 공급하고, 통신 및 다운로드도 할 수 있게 되어 있다. 저항설정은 왼쪽 그림처럼 R1, R9 저항이 왼쪽으로 0오옴이 배치된다.




[그림 2.2.8] USB 단자 용도

## 7) TFTF LCD

박막 트랜지스터 액정 디스플레이 (TFT-LCD; Thin Film Transistor-Liquid Crystal Display)는 박막 트랜지스터 (TFT)기술을 이용하여 화질을 향상시킨 액정 디스플레이 (LCD)이다.
액정 셀은 끝이 봉해진 2장의 유리판 사이에 액정이 들어 있는 구조로 유리판 내면에는 각각 나타내고자 하는 상을 표시하기 위한 전극이 형성돼 있다. 이들 전극들은 외부 단자에 전기적으로 접속돼 있다.

### (1) TFT LCD 핀 배치

STM32F103과 TFT LCD의 8비트 서로 선 연결 배치이다.

| PC0 | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 |
|---|---|---|---|---|---|---|---|
| LCD_D8 | LCD_D9 | LCD_D10 | LCD_D11 | LCD_D12 | LCD_D13 | LCD_D14 | LCD_D15 |

| PA1 | PA2 | PC8 | PC9 |
|---|---|---|---|
| LCD_RS | LCD_WR | LCD_RST | LCD_RD |

PCB 별도로 있는데 이것은 TFT LCD 와 연결이 되어 있다. 사용하지 않을 시는 GND에 연결한다.

| LCD_D0 | LCD_D1 | LCD_D2 | LCD_D3 | LCD_D4 | LCD_D5 | LCD_D6 | LCD_D7 |
|---|---|---|---|---|---|---|---|

16비트 와 8비트를 테스트 할 수 있고, 여기서는 8비트로 프로그램으로 되었다. 16비트를 테스트 할 때는 J11을 사용 하고, R23~R28 저항을 가지고 설정을 해야 한다.

- 8비트 / 16비트 설정 (0오옴 저항으로 설정)

| IM0 | IM1 | |
|---|---|---|
| 0 | 0 | =>16 Bit |
| 1 | 1 | => 8 Bit |

| IM0 | IM1 | IM2 | |
|---|---|---|---|
| 0 | 1 | 0 | =>16 Bit |
| 1 | 1 | 0 | => 8 Bit |

| IM0 | |
|---|---|
| 0 | =>16 Bit |
| 1 | =>8 Bit |

**1.I2812-7IPT2432A** 2. COM26T2844 3. TG028HBZ43

### (2) TFT LCD SPECIFICATION.

| Item | **I2812-7IPT2432A** | COM26T2844 | TG028HBZ43 | Unit |
|---|---|---|---|---|
| Display Size | **2.83** | 2.6 | 2.8 | inch |
| Module Dimension | **69.2(W)*50(H)*3.7(T)** | 70.05(W)*40.74(H)*4.78(T) | 69.2(W)*48.4(H)*3.15(T) | mm |
| Active Area | **43.2(W)*57.6(H)** | 57.40(W)*34.84(H) | 43.2(W)*57.6(H) | mm |
| Number of Dots | **176RGB*220Dots** | 240RGB*400Dots | 240RGB*320Dots | Dot |
| Driver | **ILI9328** | R61404 | HX8347-G | |
| Various Color Display | **262K** | 262K | 262K | |
| Backlight Type | **4-LED parallel** | 3-LED parallel | 4-LED parallel | |

**TFTLCD 터치스크린은 유리로 되어있어 약하기 때문에 조심해서 다루어야 한다. 특히 메인보드와 서브보드 연결 시 주의한다.**

### (3) 서브 보드 사용 가능한 LCD

1.I2812-7IPT2432A(J5커넥터), 2. COM26T2844 (J6커넥터), 3. TG028HBZ43 (J8커넥터).
I2812-7IPT2432A, TG028HBZ43는 PCB 에 직접 납땜을 하게 끔 되어있다.
아래 그림은 PCB보드에 따라 테스트 가능한 TFT LCD이다



1. I2812-7IPT2432A 2. COM26T2844 3. TG028HBZ43

[그림 2.2.9] TFT LCD

(4) I2812-7IPT2432A 초기화 관련 부분

| No. | Registers Name | R/W | RS | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IR | Index Register | W | 0 | - | - | - | - | - | - | - | - | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 |
| 00h | Driver Code Read | RO | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 01h | Driver Output Control 1 | W | 1 | 0 | 0 | 0 | 0 | 0 | SM | 0 | SS | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 02h | LCD Driving Control | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | BC0 | EOR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 03h | Entry Mode | W | 1 | TRI | DFM | 0 | BGR | 0 | 0 | 0 | 0 | ORG | 0 | I/D1 | I/D0 | AM | 0 | 0 | 0 |
| 04h | Resize Control | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | RCV1 | RCV0 | 0 | 0 | RCH1 | RCH0 | 0 | 0 | RSZ1 | RSZ0 |
| 07h | Display Control 1 | W | 1 | 0 | 0 | PTDE1 | PTDE0 | 0 | 0 | 0 | BASEE | 0 | 0 | GON | DTE | CL | 0 | D1 | D0 |
| 08h | Display Control 2 | W | 1 | 0 | 0 | 0 | 0 | FP3 | FP2 | FP1 | FP0 | 0 | 0 | 0 | 0 | BP3 | BP2 | BP1 | BP0 |
| 09h | Display Control 3 | W | 1 | 0 | 0 | 0 | 0 | 0 | PTS2 | PTS1 | PTS0 | 0 | 0 | PTG1 | PTG0 | ISC3 | ISC2 | ISC1 | ISC0 |
| 0Ah | Display Control 4 | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FMARKOE | FMI2 | FMI1 | FMI0 |
| 0Ch | RGB Display Interface Control 1 | W | 1 | 0 | ENC2 | ENC1 | ENC0 | 0 | 0 | 0 | RM | 0 | 0 | DM1 | DM0 | 0 | 0 | RIM1 | RIM0 |
| 0Dh | Frame Maker Position | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FMP8 | FMP7 | FMP6 | FMP5 | FMP4 | FMP3 | FMP2 | FMP1 | FMP0 |
| 0Fh | RGB Display Interface Control 2 | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VSPL | HSPL | 0 | DPL | EPL |
| 10h | Power Control 1 | W | 1 | 0 | 0 | 0 | SAP | 0 | BT2 | BT1 | BT0 | APE | AP2 | AP1 | AP0 | 0 | 0 | SLP | STB |
| 11h | Power Control 2 | W | 1 | 0 | 0 | 0 | 0 | 0 | DC12 | DC11 | DC10 | 0 | DC02 | DC01 | DC00 | 0 | VC2 | VC1 | VC0 |
| 12h | Power Control 3 | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VCIRE | 0 | 0 | PON | VRH3 | VRH2 | VRH1 | VRH0 |
| 13h | Power Control 4 | W | 1 | 0 | 0 | 0 | VDV4 | VDV3 | VDV2 | VDV1 | VDV0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20h | Horizontal GRAM Address Set | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD7 | AD6 | AD5 | AD4 | AD3 | AD2 | AD1 | AD0 |
| 21h | Vertical GRAM Address Set | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD16 | AD15 | AD14 | AD13 | AD12 | AD11 | AD10 | AD9 | AD8 |
| 22h | Write Data to GRAM | W | 1 | RAM write data (WD17-0) / read data (RD17-0) bits are transferred via different data bus lines according to the selected interfaces. | | | | | | | | | | | | | | | |
| 29h | Power Control 7 | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VCM5 | VCM4 | VCM3 | VCM2 | VCM1 | VCM0 |
| 2Bh | Frame Rate and Color Control | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FRS[3] | FRS[2] | FRS[1] | FRS[0] |
| 30h | Gamma Control 1 | W | 1 | 0 | 0 | 0 | 0 | 0 | KP1[2] | KP1[1] | KP1[0] | 0 | 0 | 0 | 0 | 0 | KP0[2] | KP0[1] | KP0[0] |
| 31h | Gamma Control 2 | W | 1 | 0 | 0 | 0 | 0 | 0 | KP3[2] | KP3[1] | KP3[0] | 0 | 0 | 0 | 0 | 0 | KP2[2] | KP2[1] | KP2[0] |
| 32h | Gamma Control 3 | W | 1 | 0 | 0 | 0 | 0 | 0 | KP5[2] | KP5[1] | KP5[0] | 0 | 0 | 0 | 0 | 0 | KP4[2] | KP4[1] | KP4[0] |
| 35h | Gamma Control 4 | W | 1 | 0 | 0 | 0 | 0 | 0 | RP1[2] | RP1[1] | RP1[0] | 0 | 0 | 0 | 0 | 0 | RP0[2] | RP0[1] | RP0[0] |
| 36h | Gamma Control 5 | W | 1 | 0 | 0 | 0 | VRP1[4] | VRP1[3] | VRP1[2] | VRP1[1] | VRP1[0] | 0 | 0 | 0 | 0 | VRP0[3] | VRP0[2] | VRP0[1] | VRP0[0] |
| 37h | Gamma Control 6 | W | 1 | 0 | 0 | 0 | 0 | 0 | KN1[2] | KN1[1] | KN1[0] | 0 | 0 | 0 | 0 | 0 | KN0[2] | KN0[1] | KN0[0] |
| 38h | Gamma Control 7 | W | 1 | 0 | 0 | 0 | 0 | 0 | KN3[2] | KN3[1] | KN3[0] | 0 | 0 | 0 | 0 | 0 | KN2[2] | KN2[1] | KN2[0] |
| 39h | Gamma Control 8 | W | 1 | 0 | 0 | 0 | 0 | 0 | KN5[2] | KN5[1] | KN5[0] | 0 | 0 | 0 | 0 | 0 | KN4[2] | KN4[1] | KN4[0] |
| 3Ch | Gamma Control 9 | W | 1 | 0 | 0 | 0 | 0 | 0 | RN1[2] | RN1[1] | RN1[0] | 0 | 0 | 0 | 0 | 0 | RN0[2] | RN0[1] | RN0[0] |
| 3Dh | Gamma Control 10 | W | 1 | 0 | 0 | 0 | VRN1[4] | VRN1[3] | VRN1[2] | VRN1[1] | VRN1[0] | 0 | 0 | 0 | 0 | VRN0[3] | VRN0[2] | VRN0[1] | VRN0[0] |
| 50h | Horizontal Address Start | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | HSA7 | HSA6 | HSA5 | HSA4 | HSA3 | HSA2 | HSA1 | HSA0 |
| 51h | Horizontal Address End Position | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | HEA7 | HEA6 | HEA5 | HEA4 | HEA3 | HEA2 | HEA1 | HEA0 |
| 52h | Vertical Address Start Position | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VSA8 | VSA7 | VSA6 | VSA5 | VSA4 | VSA3 | VSA2 | VSA1 | VSA0 |
| 53h | Vertical Address End Position | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VEA8 | VEA7 | VEA6 | VEA5 | VEA4 | VEA3 | VEA2 | VEA1 | VEA0 |
| 60h | Driver Output Control 2 | W | 1 | GS | 0 | NL5 | NL4 | NL3 | NL2 | NL1 | NL0 | 0 | 0 | SCN5 | SCN4 | SCN3 | SCN2 | SCN1 | SCN0 |
| 61h | Base Image Display Control | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | NDL | VLE | REV |
| 6Ah | Vertical Scroll Control | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VL8 | VL7 | VL6 | VL5 | VL4 | VL3 | VL2 | VL1 | VL0 |
| 80h | Partial Image 1 Display Position | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTDP08 | PTDP07 | PTDP06 | PTDP05 | PTDP04 | PTDP03 | PTDP02 | PTDP01 | PTDP00 |
| 81h | Partial Image 1 Area (Start Line) | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTSA08 | PTSA07 | PTSA06 | PTSA05 | PTSA04 | PTSA03 | PTSA02 | PTSA01 | PTSA00 |
| 82h | Partial Image 1 Area (End Line) | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTEA08 | PTEA07 | PTEA06 | PTEA05 | PTEA04 | PTEA03 | PTEA02 | PTEA01 | PTEA00 |
| 83h | Partial Image 2 Display Position | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTDP18 | PTDP17 | PTDP16 | PTDP15 | PTDP14 | PTDP13 | PTDP12 | PTDP11 | PTDP10 |
| 84h | Partial Image 2 Area (Start Line) | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTSA18 | PTSA17 | PTSA16 | PTSA15 | PTSA14 | PTSA13 | PTSA12 | PTSA11 | PTSA10 |
| 85h | Partial Image 2 Area (End Line) | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTEA18 | PTEA17 | PTEA16 | PTEA15 | PTEA14 | PTEA13 | PTEA12 | PTEA11 | PTEA10 |
| 90h | Panel Interface Control 1 | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | DIVI1 | DIVI00 | 0 | 0 | 0 | 0 | RTNI3 | RTNI2 | RTNI1 | RTNI0 |
| 92h | Panel Interface Control 2 | W | 1 | 0 | 0 | 0 | 0 | 0 | NOWI2 | NOWI1 | NOWI0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 95h | Panel Interface Control 4 | W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | DIVE1 | DIVE0 | 0 | 0 | RTNE5 | RTNE4 | RTNE3 | RTNE2 | RTNE1 | RTNE0 |
| A1h | OTP VCM Programming Control | W | 1 | 0 | 0 | 0 | 0 | OTP_PGM_EN | 0 | 0 | 0 | 0 | 0 | VCM_OTP5 | VCM_OTP4 | VCM_OTP3 | VCM_OTP2 | VCM_OTP1 | VCM_OTP0 |
| A2h | OTP VCM Status and Enable | W | 1 | PGM_CNT1 | PGM_CNT0 | VCM_D5 | VCM_D4 | VCM_D3 | VCM_D2 | VCM_D1 | VCM_D0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VCM_EN |
| A5h | OTP Programming ID Key | W | 1 | KEY 15 | KEY 14 | KEY 13 | KEY 12 | KEY 11 | KEY 10 | KEY 9 | KEY 8 | KEY 7 | KEY 6 | KEY 5 | KEY 4 | KEY 3 | KEY 2 | KEY 1 | KEY 0 |

## Driver Output Control (R01h)

LCD_WR_REG(0x0001,0x0100); // set SS and SM bit

| R/W | RS | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | SM | 0 | SS | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SS:** Select the shift direction of outputs from the source driver.

When SS = 0, the shift direction of outputs is from S1 to S720

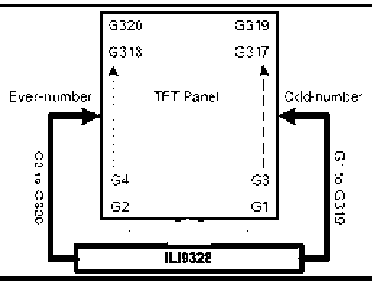
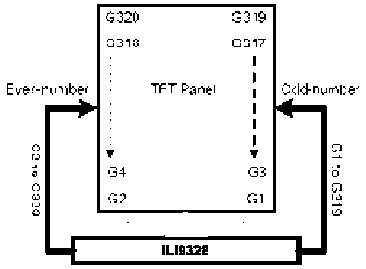
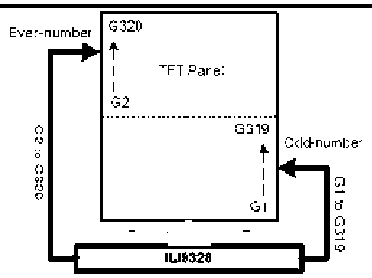
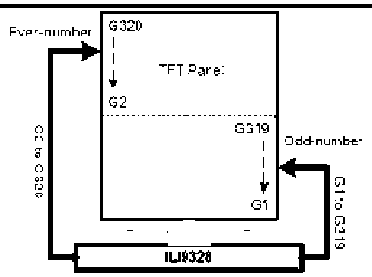
When SS = 1, the shift direction of outputs is from S720 to S1.

In addition to the shift direction, the settings for both SS and BGR bits are required to change the assignment of R, G, B dots to the source driver pins.

To assign R, G, B dots to the source driver pins from S1 to S720, set SS = 0.

To assign R, G, B dots to the source driver pins from S720 to S1, set SS = 1.

***When changing SS or BGR bits, RAM data must be rewritten.***

| SM | GS | Scan Direction | Gate Output Sequence |
|---|---|---|---|
| 0 | 0 |  | G1, G2, G3, G4, ....G316<br>G317, G318, G319, G320 |
| 0 | 1 |  | G320, G319, G318, ....<br>G6, G5, G4, G3, G2, G1 |
| 1 | 0 |  | G1, G3, G5, G7, ....G311<br>G313, G315 G317, G319<br><br>G2, G4, G6, G8, ....G312<br>G314, G316, G318, G320 |
| 1 | 1 |  | G320, G318, G316, ....<br>G10, G8, G6, G4, G2<br><br>G319, G317, G315, ....<br>G9, G78, G5, G3, G1 |

**Entry Mode (R03h)**

GRAM을 액세스 할 때 1픽셀 디스플레이 증가, 감소한다.
BGR = "0": 픽셀 데이터를 쓸 수 RGB 순서를 따른다.
BGR = "1": GRAM에 BGR에 RGB 데이터를 스왑.

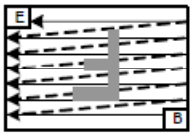
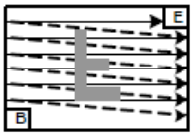
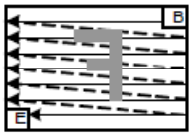
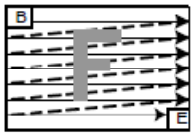
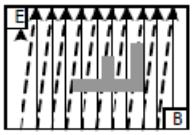
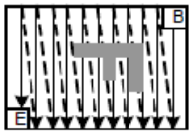
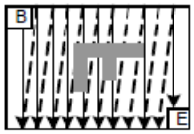
| R/W | RS | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | TRI | DFM | 0 | BGR | 0 | 0 | 0 | 0 | ORG | 0 | I/D1 | I/D0 | AM | 0 | 0 | 0 |

**AM** Control the GRAM update direction.

When AM = "0", the address is updated in horizontal writing direction.

When AM = "1", the address is updated in vertical writing direction.

When a window area is set by registers R50h ~R53h, only the addressed GRAM area is updated based on I/D[1:0] and AM bits setting.

| | I/D[1:0] = 00<br>Horizontal : decrement<br>Vertical : decrement | I/D[1:0] = 01<br>Horizontal : increment<br>Vertical : decrement | I/D[1:0] = 10<br>Horizontal : decrement<br>Vertical : increment | I/D[1:0] = 11<br>Horizontal : increment<br>Vertical : increment |
|---|---|---|---|---|
| AM = 0<br>Horizontal |  |  |  |  |
| AM = 1<br>Vertical |  |  |  |  |

**16-bit System Interface Data Format (16-bit 262,144 컬러와 65,536 컬러 표시한다. )**

**8-bit System Interface Data Format ( 262,144 컬러와 65,536 컬러 표시한다. )**

## Display Control 1 (R07h)

| R/W | RS | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | PTDE1 | PTDE0 | 0 | 0 | 0 | BASEE | 0 | 0 | GON | DTE | CL | 0 | D1 | D0 |

**D[1:0]** Set D[1:0]="11" to turn on the display panel, and D[1:0]="00" to turn off the display panel.

| D1 | D0 | BASEE | Source, VCOM Output | ILI9328 internal operation |
|---|---|---|---|---|
| 0 | 0 | 0 | GND | Halt |
| 0 | 1 | 1 | GND | Operate |
| 1 | 0 | 0 | Non-lit display | Operate |
| 1 | 1 | 0 | Non-lit display | Operate |
| **1** | **1** | **1** | **Base image display** | **Operate** |

| CL | Colors |
|---|---|
| 0 | 262,144 |
| 1 | 8 |

## RGB Display Interface Control 1 (R0Ch)

| R/W | RS | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | ENC2 | ENC1 | ENC0 | 0 | 0 | 0 | RM | 0 | 0 | DM1 | DM0 | 0 | 0 | RIM1 | RIM0 |

**RIM[1:0]** Select the RGB interface data width.

| RIM1 | RIM0 | RGB Interface Mode |
|---|---|---|
| 0 | 0 | 18-bit RGB interface (1 transfer/pixel), DB[17:0] |
| 0 | 1 | 16-bit RGB interface (1 transfer/pixel), DB[17:13] and DB[11:1] |
| 1 | 0 | 6-bit RGB interface (3 transfers/pixel), DB[17:12] |
| 1 | 1 | Setting disabled |

*Note1: Registers are set only by the system interface.*

*Note2: Be sure that one pixel (3 dots) data transfer finished when interface switch.*

| DM1 | DM0 | Display Interface |
|---|---|---|
| 0 | 0 | Internal system clock |
| 0 | 1 | RGB interface |
| 1 | 0 | VSYNC interface |
| 1 | 1 | Setting disabled |

### (5) HX8347G 초기화

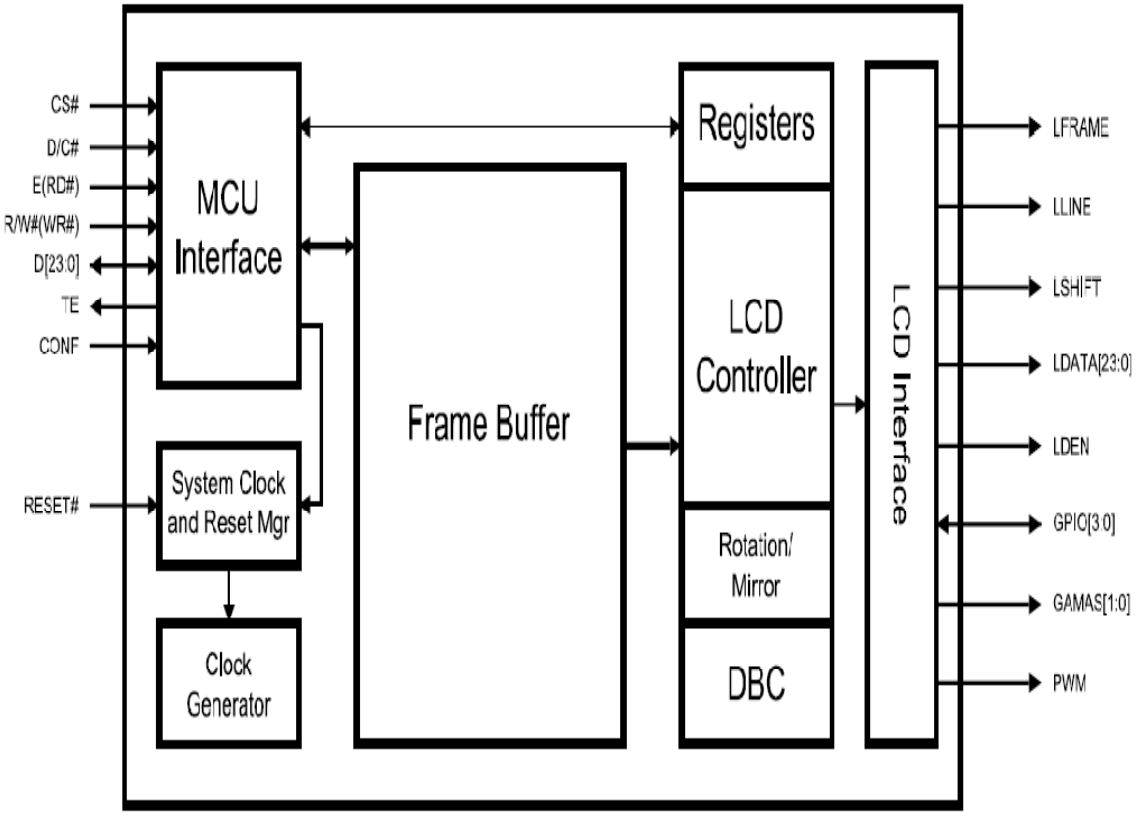
| (Hex) | Operation Code | W/R | Upper Code | Lower Code | | | | | | | | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | D[17:8] | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | |
| 00 | Himax ID | R | - | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | - |
| 01 | Display Mode control | W/R | - | DP_STB(0) | DP_STBS(0) | - | - | SCROL(0) | IDMON(0) | INVON(0) | PTLON(0) | - |
| 02 | Column address start 2 | W/R | - | SC[15:8] (8'b0000_0000) | | | | | | | | - |
| 03 | Column address start 1 | W/R | - | SC[7:0] (8'b0000_0000) | | | | | | | | - |
| 04 | Column address end 2 | W/R | - | EC[15:8] (8'b0000_0000) | | | | | | | | - |
| 05 | Column address end 1 | W/R | - | EC[7:0] (8'b1110_1111) | | | | | | | | - |
| 06 | Row address start 2 | W/R | - | SP[15:8] (8'b0000_0000) | | | | | | | | - |
| 07 | Row address start 1 | W/R | - | SP[7:0] (8'b0000_00000) | | | | | | | | - |
| 08 | Row address end 2 | W/R | - | EP[15:8] (8'b0000_0001) | | | | | | | | - |
| 09 | Row address end 1 | W/R | - | EP[7:0] (8'b0011_1111) | | | | | | | | - |
| 0A | Partial area start row 2 | W/R | - | PSL[15:8] (8'b0000_0000) | | | | | | | | - |
| 0B | Partial area start row 1 | W/R | - | PSL[7:0] (8'b0000_00000) | | | | | | | | - |
| 0C | Partial area end row 2 | W/R | - | PEL[15:8] (8'b0000_0001) | | | | | | | | - |
| 0D | Partial area end row 1 | W/R | - | PEL[7:0] (8'b0011_1111) | | | | | | | | - |
| 0E | Vertical Scroll Top fixed area 2 | W/R | - | TFA[15:8] (8'b0000_0000) | | | | | | | | - |
| 0F | Vertical Scroll Top fixed area 1 | W/R | - | TFA[7:0] (8'b0000_0000) | | | | | | | | - |
| 10 | Vertical Scroll height area 2 | W/R | - | VSA[15:8] (8'b0000_0001) | | | | | | | | - |
| 11 | Vertical Scroll height area 1 | W/R | - | VSA[7:0] (8'b0100_0000) | | | | | | | | - |
| 12 | Vertical Scroll Button area 2 | W/R | - | BFA[15:8] (8'b0000_0000) | | | | | | | | - |
| 13 | Vertical Scroll Button area 1 | W/R | - | BFA [7:0] (8'b0000_0000) | | | | | | | | - |
| 14 | Vertical Scroll Start address 2 | W/R | - | VSP [15:8] (8'b0000_0000) | | | | | | | | - |
| 15 | Vertical Scroll Start address 1 | W/R | - | VSP [7:0] (8'b0000_0000) | | | | | | | | - |
| 16 | Memory Access control | W/R | - | MY(0) | MX(0) | MV(0) | ML(0) | BGR(0) | - | - | - | - |
| 17 | COLMOD | W/R | - | CSEL[3:0] (4b'0110) | | | | - | IFPF[2:0] (3b'110) | | | - |
| 18 | OSC Control 2 | W/R | - | I/PI_RADJ1[3:0] (3b'0011) | | | | N/P_RADJ0[3:0](4b'0100) | | | | - |
| 19 | OSC Control 1 | W/R | - | - | - | - | - | - | - | - | OSC_EN(0) | - |
| 1A | Power Control 1 | W/R | - | - | - | - | - | - | BT[2:0] (001) | | | - |
| 1B | Power Control 2 | W/R | - | - | - | VRH[5:0] (01_1011)_4.8V | | | | | | - |
| 1C | Power Control 3 | W/R | - | - | - | - | - | - | AP[2:0] (011) | | | - |
| 1D | Power Control 4 | W/R | - | - | I/PI_FS0[2:0](100) | | | - | N/P_FS0[2:0] ](100) | | | - |
| 1E | Power Control 5 | W/R | - | - | I/PI_FS1[2:0] ](100) | | | - | N/P_FS1[2:0] ](100) | | | - |
| 1F | Power Control 6 | W/R | - | GASEN(1) | VCOMG(0) | - | PON(0) | DK(1) | XDK(0) | DDVDH_TRI(0) | STB(1) | - |
| 22 | SRAM Write Control | W/R | SRAM Write | | | | | | | | | - |
| 23 | VCOM Control 1 | W/R | - | VMF[7:0](1000_0000) | | | | | | | | - |
| 24 | VCOM Control 2 | W/R | - | VMH[7:0](0010_1111) | | | | | | | | - |
| 25 | VCOM Control 3 | W/R | - | VML[7:0](0101_0111) | | | | | | | | - |
| 26 | Display Control 1 | W/R | - | -- | - | - | - | ISC[3:0](0001) | | | | - |
| 27 | Display Control 2 | W/R | - | PT[1:0](10) | | PTV[1:0](10) | | - | - | PTG(1) | REF(1) | - |

| (Hex) | Operation Code | W/R | Upper Code | Lower Code | | | | | | | | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | D[17:8] | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | |
| 28 | Display Control 3 | W/R | - | - | - | GON(1) | DTE(0) | D[1:0] (00) | | - | - | - |
| 29 | Frame Rate control 1 | W/R | - | I/PI_RTN[3:0](1000) | | | | N/P_RTN[3:0](1000) | | | | - |
| 2A | Frame Rate Control 2 | W/R | - | - | - | I/PI_DIV[1:0](00) | | - | - | N/P_DIV[1:0](00) | | - |
| 2B | Frame Rate Control 3 | W/R | - | N/P_DUM[7:0] (8'b0001_1100) | | | | | | | | - |
| 2C | Frame Rate Control 4 | W/R | - | I/PI_DUM[7:0] (8'b0001_1100) | | | | | | | | - |
| 2D | Cycle Control 1 | W/R | - | GDON[7:0] (8'b0000_1101) | | | | | | | | - |
| 2E | Cycle Control 2 | W/R | - | GDOF[7:0] (8'b0111_1000) | | | | | | | | - |
| 2F | Display inversion | W/R | - | - | I/PI_NW[2:0](3b'001) | | | - | N/P_NW[2:0] (3b'001) | | | - |
| 31 | RGB interface control 1 | W/R | - | - | - | - | - | - | - | RCM[1:0](00) | | - |
| 32 | RGB interface control 2 | W/R | - | - | - | - | - | DPL (0) | HSPL (0) | VSPL (0) | EPL (0) | - |
| 33 | RGB interface control 3 | W/R | - | HBP[7:0] | | | | | | | | - |
| 34 | RGB interface control 4 | W/R | - | HBP[9:8] | | VBP[5:0] | | | | | | - |
| 36 | Panel Characteristic | W/R | - | - | - | - | - | SS_Panel | GS_Panel | REV_Panel | BGR_Panel | - |
| 38 | OTP Control 1 | W/R | - | OTP_PTM[1:0] | | OTP_VARDJ[1:0] | | OTP_POR | OTP_OTPEN | OTP_PROG | OTP_PWE | - |
| 39 | OTP Control 2 | W/R | - | - | - | - | - | - | OTP_YA2 | OTP_YA1 | OTP_YA0 | - |
| 3A | OTP Control 3 | W/R | - | - | - | - | OTP_XA4 | OTP_XA3 | OTP_XA2 | OTP_XA1 | OTP_XA0 | - |
| 3B | OTP Control 4 | R | | OTPDATA7 | OTPDATA6 | OTPDATA5 | OTPDATA4 | OTPDATA3 | OTPDATA2 | OTPDATA1 | OTPDATA0 | |
| 3C | CABC Control 1 | W/R | - | DBV[7:0](8'h00) | | | | | | | | - |
| 3D | CABC Control 2 | W/R | - | - | - | BCTRL (0) | - | DD (0) | BL (0) | - | - | - |
| 3E | CABC Control 3 | W/R | - | - | - | - | - | - | - | C1 (0) | C0 (0) | - |
| 3F | CABC Control 4 | W/R | - | CMB[7:0](8'h00) | | | | | | | | - |
| 40 | r1 Control (1) | W/R | - | - | - | VRP0[5:0] | | | | | | - |
| 41 | r1 Control (2) | W/R | - | - | - | VRP1[5:0] | | | | | | - |
| 42 | r1 Control (3) | W/R | - | - | - | VRP2[5:0] | | | | | | - |
| 43 | r1 Control (4) | W/R | - | - | - | VRP3[5:0] | | | | | | - |
| 44 | r1 Control (5) | W/R | - | - | - | VRP4[5:0] | | | | | | - |
| 45 | r1 Control (6) | W/R | - | - | - | VRP5[5:0] | | | | | | - |
| 46 | r1 Control (7) | W/R | - | - | PRP0[6:0] | | | | | | | - |
| 47 | r1 Control (8) | W/R | - | - | PRP1[6:0] | | | | | | | - |
| 48 | r1 Control (9) | W/R | - | - | - | - | PKP0[4:0] | | | | | - |
| 49 | r1 Control (10) | W/R | - | - | - | - | PKP1[4:0] | | | | | - |
| 4A | r1 Control (11) | W/R | - | - | - | - | PKP2[4:0] | | | | | - |
| 4B | r1 Control (12) | W/R | - | - | - | - | PKP3[4:0] | | | | | - |
| 4C | r1 Control (13) | W/R | - | - | - | - | PKP4[4:0] | | | | | - |
| 50 | r1 Control (14) | W/R | - | - | - | VRN0[5:0] | | | | | | - |
| 51 | r1 Control (15) | W/R | - | - | - | VRN1[5:0] | | | | | | - |
| 52 | r1 Control (16) | W/R | - | - | - | VRN2[5:0] | | | | | | - |
| 53 | r1 Control (17) | W/R | - | - | - | VRN3[5:0] | | | | | | - |
| 54 | r1 Control (18) | W/R | - | - | - | VRN4[5:0] | | | | | | - |
| 55 | r1 Control (19) | W/R | - | - | - | VRN5[5:0] | | | | | | - |
| 56 | r1 Control (20) | W/R | - | - | PRN0[6:0] | | | | | | | - |
| 57 | r1 Control (21) | W/R | - | - | PRN1[6:0] | | | | | | | - |
| 58 | r1 Control (22) | W/R | - | - | - | PKN0[4:0] | | | | | | - |
| 59 | r1 Control (23) | W/R | - | - | - | PKN1[4:0] | | | | | | - |
| 5A | r1 Control (24) | W/R | - | - | - | PKN2[4:0] | | | | | | - |
| 5B | r1 Control (25) | W/R | - | - | - | PKN3[4:0] | | | | | | - |
| 5C | r1 Control (26) | W/R | - | - | - | PKN4[4:0] | | | | | | - |
| 5D | r1 Control (27) | W/R | - | CGMN1[1:0] | | CGMN0[1:0] | | CGMP1[1:0] | | CGMP0[1:0] | | - |
| 60 | TE Control | W/R | - | - | - | | TE_mode(0) | TEOE(0) | | - | - | - |
| 61 | ID1 | W/R | - | ID17 | ID16 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | - |
| 62 | ID2 | W/R | - | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | - |
| 63 | ID3 | W/R | - | ID37 | ID36 | ID35 | ID34 | ID33 | ID32 | ID31 | ID30 | - |

| (Hex) | Operation Code | W/R | Upper Code | Lower Code | | | | | | | | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | D[17:8] | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | |
| 84 | TE Output line2 | W/R | - | TESEL15 | TESEL14 | TESEL 13 | TESEL 12 | TESEL11 | TESEL10 | TESEL9 | TESEL8 | - |
| 85 | TE Output line1 | W/R | - | TESEL 7 | TESEL 6 | TESEL 5 | TESEL 4 | TESEL 3 | TESEL 2 | TESEL1 | TESEL 0 | - |
| E4 | Power saving 1 | W/R | - | EQ_S1[7:0] | | | | | | | | - |
| E5 | Power saving 2 | W/R | - | EQ_S2[7:0] | | | | | | | | - |
| E6 | Power saving 3 | W/R | - | EQ_S3[7:0] | | | | | | | | - |
| E7 | Power saving 4 | W/R | - | EQ_S4[7:0] | | | | | | | | - |
| E8 | Source OP control_Normal | W/R | - | OPON_N[7:0] | | | | | | | | - |
| E9 | Source OP control_IDLE | W/R | - | OPON_I[7:0] | | | | | | | | - |
| EA | Power control internal use (1) | W/R | - | STBA[15:8] | | | | | | | | - |
| EB | Power control internal use (2) | W/R | - | STBA[7:0] | | | | | | | | - |
| EC | Source control internal use (1) | W/R | - | PTBA[15:8] | | | | | | | | - |
| ED | Source control internal use (2) | W/R | - | PTBA[7:0] | | | | | | | | - |
| FF | Page select | W/R | - | - | - | - | - | - | - | PAGE_SEL[1:0] (00) | | - |

**(6) SSD1963 초기화**

LCD 판넬에 LCD드라이브가 없는 판넬에 직접 부착해 컨트롤 한다. 위에 3가지가 각각 다르듯이 SSD1963 역시 초기화를 해 주어야 한다. 이 부분은 서브보드 별도로 제작되어 있다. LCD 사이즈3인치에서 7인치 최대 864 x 480까지 사용할 때 많이 사용한다.



LCD 판넬에 LCD드라이브가 없는 판넬에 직접 부착해 컨트롤 한다. 위에 3가지가 각각 다르듯이 SSD1963 역시 초기화를 해 주어야 한다. 이 부분은 서브보드 별도로 제작되어 있다.

RESET(127핀) =>H
CONF(128핀) =>H (8080인터페이스를 사용한다.)
PLL어드레스 (0xE0) 를 0x01 enable 한 후 0x03을 하여 start 한다. PLL Frequency 어드레스 (0xE2)는 10MHz*36/3 = 120MHz 에서 36과 360Mhz = 10Mhz x (35+1), 120Mhz = 360Mhz / 3 에서 3 나온다. 여기서 각각 1을 빼면 35, 2 가 나온다. 그래서 PLL Frequency 어드레스 (0x )는 0x , 0x02 를 쓰고, 0x54 더미를 추가 써 준다. 여기서는 10Mhz 를 사용하였지만, 8MHz를 사용하면 계산 값이 달라진다.

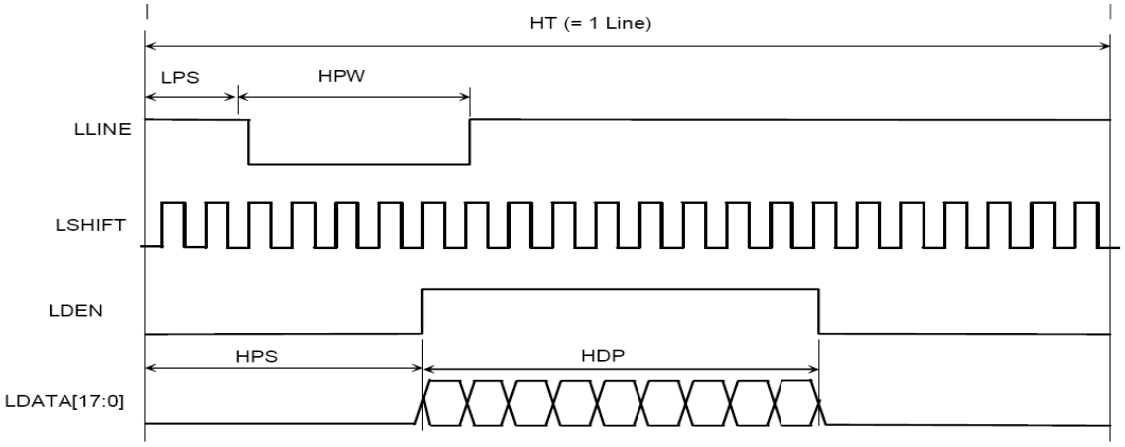
여기서는 삼성의 LMS700KF06을 예를 들어 설명한다.

**Vertical timing**

| Signal | Symbol | Min. | Typ. | Max. | Unit | Note |
|---|---|---|---|---|---|---|
| Frame Frequency | fFRM | - | 60 | - | Hz | |
| Vertical Back porch | VBP | - | 8 | - | H | VPS |
| Vertical Front porch | VFP | - | 5 | - | H | *Note |

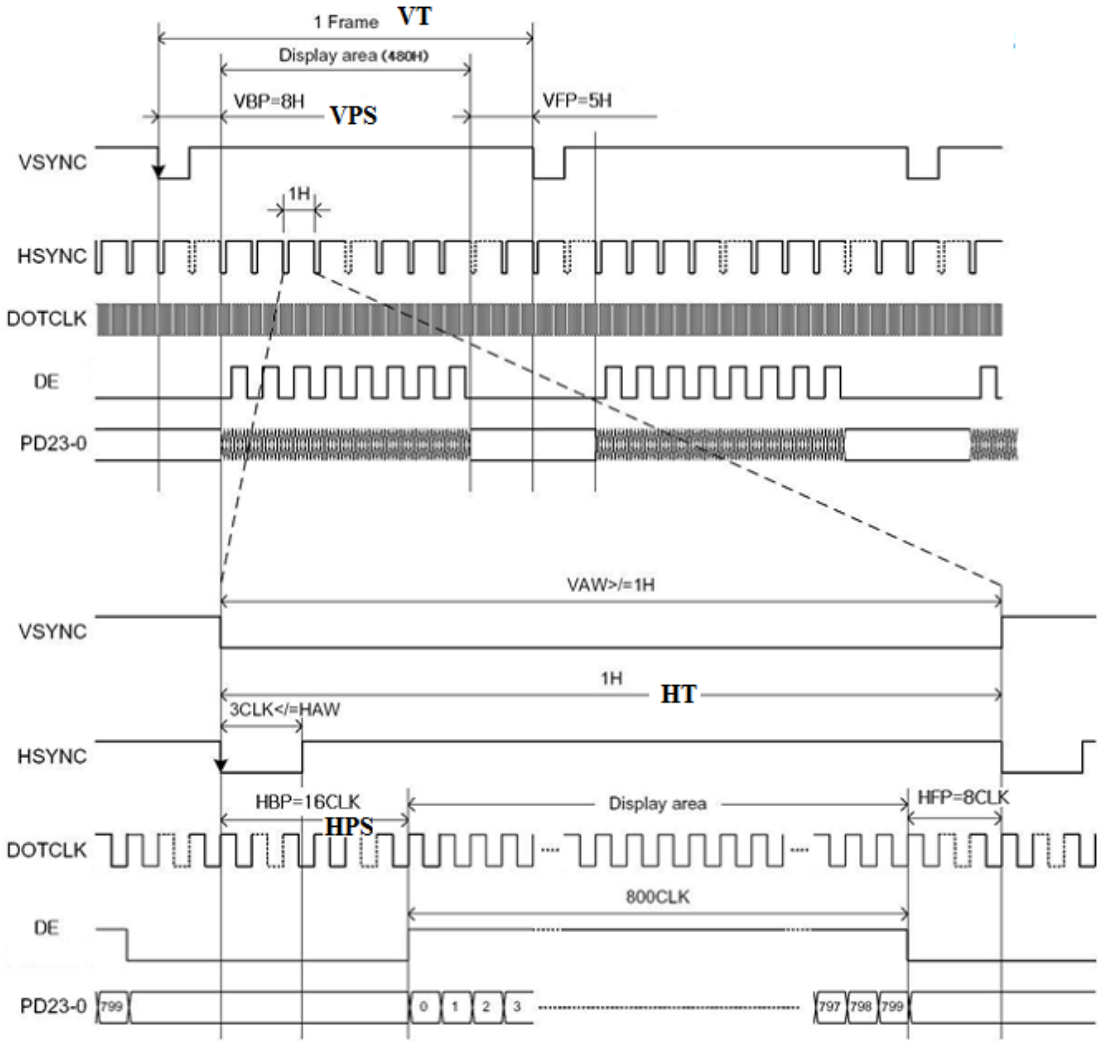
**Horizontal timing**

| Signal | Symbol | Min. | Typ. | Max. | Unit | Note |
|---|---|---|---|---|---|---|
| Horizontal Back porch | HBP | - | 16 | - | DOTCLK | HPS |
| Horizontal Front porch | HFP | - | 8 | - | DOTCLK | *Note |
| DOTCLK Frequence | $f_{DOTCLK}$ | - | 24.5 | - | MHz | @fFRM=60Hz |



DOTCLK 이 24.5MHz = (120MHz * (LCDC_FPR+ 1)) / 2^20 에서 TFTLCD 마다 다르므로 실제 코딩하면서 셋팅을 해야 한다.

**Interface Timing**




```
HT      823  // (HPS(HPW+HBP)+HDP+HFP)=824=16+800+8  //Horizontal Total
HDP     799  //  HDP
LPS       0  //  Horizontal Display Period Start Position
HPS      15  // 16-1 (HPW+HBP)  //LLINE Pulse Start Position
HPW       3  //  4-1 LLINE Pulse Width

VT      493 //  (VPS(또는 VPW+VBP)+VDP+VFP) =8+480+5  //Vertical Total
VDP     479 // 480-1
FPS       0 // Vertical Display Period Start Positio
VPS       7 // 8-1  (VPW+ VBP)  // LFRAME Pulse Start Position  => 8+1=>9를 해야 함
VPW       0 // 10-1  LFRAME Pulse Width
```

LCDC_FPR 은 214083 이고 이를 HEX코드로로 전환 하면0x034443 이 된다.
LCD_PIXEL_CLOCK의 어드레스 (0x00E6) 에 쓰면 된다.

LCD Mode (B0H, B1H) =>7번째 파라메타
G[5:3] : Even line RGB sequence (POR = 000)
000 RGB
101 BGR

G[2:1] : Odd line RGB sequence (POR = 000)
000 RGB
101 BGR

- 백라이트 설정

Command BEh

| | D/C | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Hex |
|---|---|---|---|---|---|---|---|---|---|---|
| Command | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | BE |
| Parameter 1 | 1 | $PWMF_7$ | $PWMF_6$ | $PWMF_5$ | $PWMF_4$ | $PWMF_3$ | $PWMF_2$ | $PWMF_1$ | $PWMF_0$ | xx |
| Parameter 2 | 1 | $PWM_7$ | $PWM_6$ | $PWM_5$ | $PWM_4$ | $PWM_3$ | $PWM_2$ | $PWM_1$ | $PWM_0$ | xx |
| Parameter 3 | 1 | 0 | 0 | 0 | 0 | $C_3$ | 0 | 0 | $C_0$ | xx |
| Parameter 4 | 1 | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | xx |
| Parameter 5 | 1 | $E_7$ | $E_6$ | $E_5$ | $E_4$ | $E_3$ | $E_2$ | $E_1$ | $E_0$ | xx |
| Parameter 6 | 1 | 0 | 0 | 0 | 0 | $F_3$ | $F_2$ | $F_1$ | $F_0$ | xx |

PWMF[7:0] : Set the PWM frequency in system clock (POR = 00000000)

PWM signal frequency = PLL clock / (256 * PWMF[7:0]) / 256

PWM[7:0] : Set the PWM duty cycle (POR = 00000000) PWM duty cycle = PWM[7:0] / 256
Note : PWM always 0 if PWM[7:0] = 00h

CAT4238 Datasheet 일부

| Symbol | Parameter | Test Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $I_Q$ | Operating Current | $V_{FB}$ = 0.2 V<br>$V_{FB}$ = 0.4 V (not switching) | | 0.6<br>0.1 | 1.5<br>0.6 | mA |
| $I_{SD}$ | Shutdown Current | $V_{\overline{SHDN}}$ = 0 V | | 0.1 | 1 | µA |
| $V_{FB}$ | FB Pin Voltage | 10 LEDs with $I_{LED}$ = 20 mA | 285 | 300 | 315 | mV |
| $I_{FB}$ | FB pin input leakage | | | | 1 | µA |
| $I_{LED}$ | Programmed LED Current | R1 = 10 Ω<br>R1 = 15 Ω<br>R1 = 20 Ω | 28.5<br>19<br>14.25 | 30<br>20<br>15 | 31.5<br>21<br>15.75 | mA |
| $V_{IH}$<br>$V_{IL}$ | $\overline{SHDN}$ Logic High<br>$\overline{SHDN}$ Logic Low | Enable Threshold Level<br>Shutdown Threshold Level | <br>0.4 | 0.8<br>0.7 | 1.5 | V |
| $F_{SW}$ | Switching Frequency | | 0.8 | 1.0 | 1.3 | MHz |

CAT4238 사용시 2를 사용한다.

1 일때 120Mhz /(256 * 1)/256 = 1.831MHz

2 일때 120Mhz /(256 * 2)/256 = 915.5KHz => 데이터시트를 보면 1MHz 에 가깝기 때문에

3 일때 120Mhz /(256 * 3)/256 = 610.3KHz

```
LCD_WR_REG(0X00BE);
LCD_WR_Data(0x0002); //2 일때 120Mhz /(256 * 2)/256 = 915.5KHz
LCD_WR_Data(val);  // LCD 밝기 조절 0~255
LCD_WR_Data(0x0001);
LCD_WR_Data(0x00FF);
LCD_WR_Data(0x0000);
LCD_WR_Data(0x0000);
```

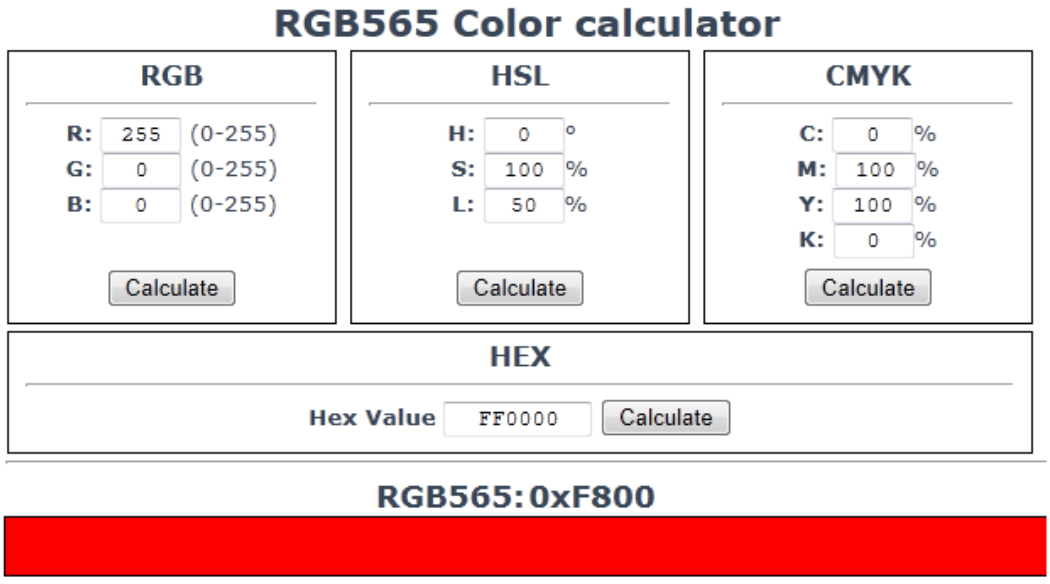
- RGB 565 Format 프로그램

TFTLCD 연결을 RGB 565 포맷으로 하였기 때문에 적색을 표시할 때 0XF800 으로 나타낸다.

#define RED  0XF800

이 부분을 자동으로 계산 프로그램 사이트는 아래사이트를 참고한다.

http://www.henningkarlsen.com/electronics/calc_rgb565.php



## - PCX 파일 디스플레이

PCX는 Z소프트에서 개발한 그래픽 파일 포맷이다. Z소프트가 개발한 도스기반의 페인트브러시 프로그램에서 사용하기 위하여 만들어졌다. 최근에는 더 나은 사양으로 인해 거의 사용하지 않고 있으나, 여기서는 800x480 이미지 사이즈를 디스플레이를 BMP 파일을 적용이 안되기 때문에 PCX 이미지를 적용을 하였다.

TFTLCD에 올릴 이미지를 편집한다. 800x480, 320x240 이미지를 각각 디스플레이 한다.

800x480 이미지는 2개는 STM32F407 프로그램 사이즈로 못 올리기 때문이다.

먼저 800x480 로 편집한 800x480 (1.09MB) 사이즈 BMP 그림파일이 용량이 크기 때문에 PCX 변환한다. 알씨 프로그램을 사용하여 320KB로 변환한다.

Pcxconv.exe 를 사용하여 헥사코드로 변환한다. 아래 그림은 PCX 파일을 .C 로 변환한 과정이다.

C 파일을 열어 보면

```
const char UserPCX[]={
0x20,0x03,0xE0,0x01,
 ……( 생략)
0xC5,0x00
};
```

이 부분 전체를 프로그램에 넣어 불러 오면 된다.

```
void lcd_pic(unsigned short x,unsigned short y, const unsigned char *src)

 unsigned short i,j;
 unsigned short w,h,rgb[256];
 unsigned char i1,i2,ch,rc;
 int c,x2,y2;

 x2 = y; y2 = x;
 w=(*(src))+(*(src+1)<<8);
 h=(*(src+2))+(*(src+3)<<8);

 LCD_WR_REG(0x002A);
 LCD_WR_Data(x>>8);
 LCD_WR_Data(x&0x00ff);
 LCD_WR_Data((x+w-1)>>8);
 LCD_WR_Data((x+w-1)&0x00ff);
 LCD_WR_REG(0x002b);
 LCD_WR_Data(y>>8);
 LCD_WR_Data(y&0x00ff);
 LCD_WR_Data((y+h-1)>>8);
 LCD_WR_Data((y+h-1)&0x00ff);
 LCD_WR_REG(0x002c);

 for (i=0;i<256;i++){
        rgb[i]= *(src+4+(i*2)) + (*(src+5+(i*2))<<8);
 }

   c = 0;
 for(j=0; j<h; j++) {

    //TFT_GRAM_address(x2+j,y2);

    for(i= 0; i<w;) {

       ch = *(src+(4+(256*2))+c);
       if((ch & 0xc0) == 0xc0) {
           rc = (ch & 0x3f);
           c++;
           while(rc--) {
              LCD_WR_Data( rgb[*(src+(4+(256*2))+c)] );
                                 i++;
           }
                        c++;
       }
       else {
           LCD_WR_Data( rgb[ch] );
           i++; c++;
       }
    }
```

```
}

이미지 불러 오기

 lcd_pic(0,0,UserPCX);
```

이와 같은 방법으로 다른 이미지 파일도 TFTLCD 올릴 수 있다.

참고로 800x480 이미지는 1.09MB 사이즈 BMP 그림파일이 파일 c로 변환된 사이즈는 3.75MB 이고, PCX 이미지는 320KB에서 c로 변환된 1.6MB 이다.

### - BMP 파일 디스플레이

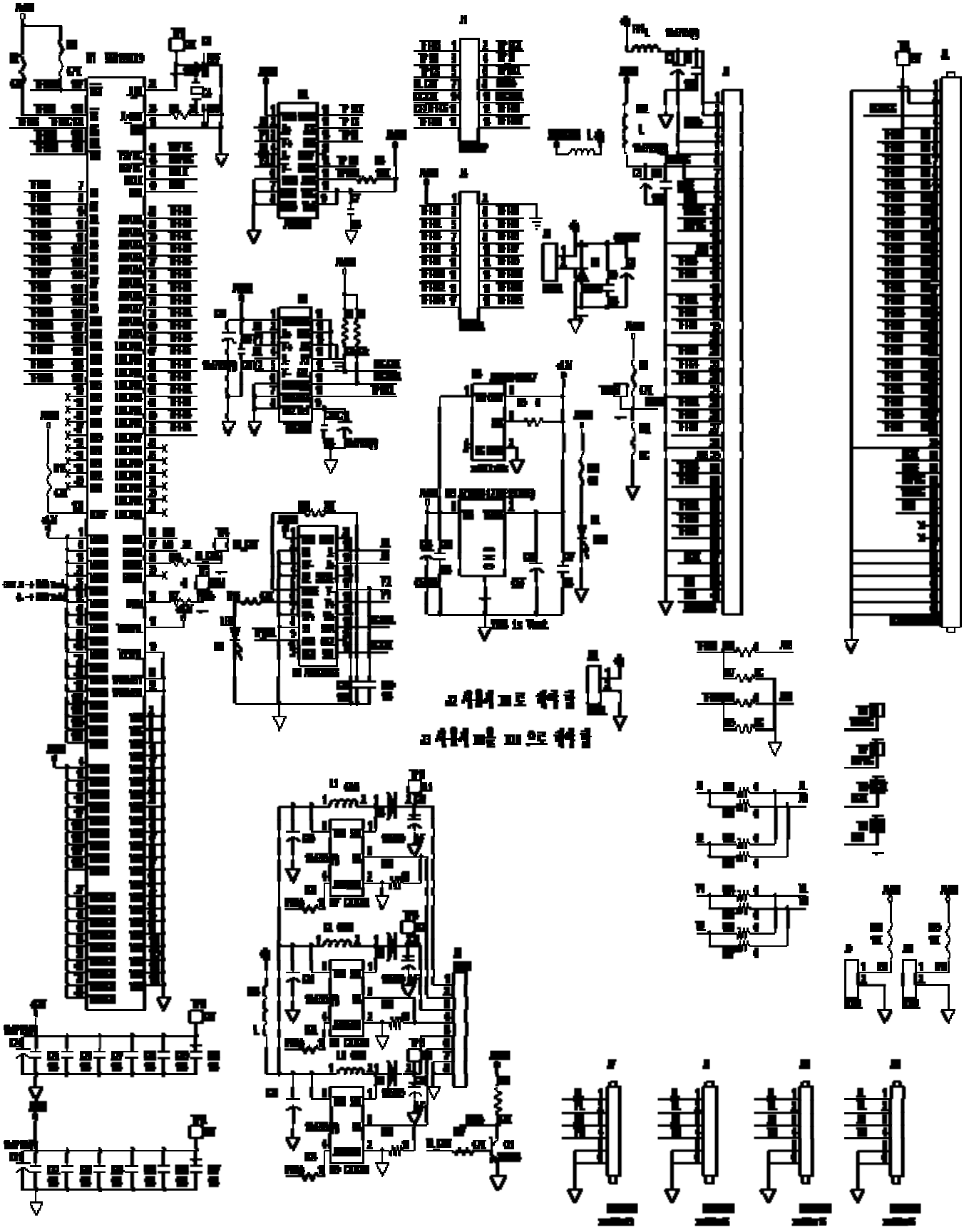
작은 이미지의 BMP 파일을 디스플레이 하는 과정이다..

픽셀 59x59 사이즈 10.5kB 를 c코드로 변환하면 34.9kB 로 된다.

변환프로그램 (Image2Lcd )을 사용하여 c 코드로 변환하여 이미지를 올린다.

```
void TFT_Disp_sys_ico(unsigned int xsta,unsigned int ysta,unsigned int
xend,unsigned int yend,unsigned char picturenumber)
{
  unsigned long n,disp_pix_num;
  unsigned int temp;
    disp_pix_num=(unsigned long)(yend-ysta)*(xend-xsta)*2;
LCD_WR_REG(0x002A);
LCD_WR_Data(xsta>>8);
LCD_WR_Data(xsta&0x00ff);
LCD_WR_Data(xend-1>>8);
LCD_WR_Data(xend-1&0x00ff);
          LCD_WR_REG(0x002b);
LCD_WR_Data(ysta>>8);
LCD_WR_Data(ysta&0x00ff);
LCD_WR_Data(yend-1>>8);
LCD_WR_Data(yend-1&0x00ff);
LCD_WR_REG(0x002c);
n=0;
     while(n<disp_pix_num)
 {
   temp=(uint16_t)(Systerm_ico_XX[n+picturenumber*
   6970]<<8)+Systerm_ico_XX[n+1+picturenumber*6970];
       LCD_WR_Data(temp);//
       n=n+2;
      }
//이미지 실행
TFT_Disp_sys_ico(100,100,100+59,100+60,0); //예제 이미지
```

```
LCD_WR_REG(0x002A);
LCD_WR_Data(xChar>>8);
LCD_WR_Data(xChar&0x00ff);
LCD_WR_Data(xChar+7>>8);
LCD_WR_Data(xChar+7&0x00ff);
LCD_WR_REG(0x002b);
LCD_WR_Data(yChar>>8);
LCD_WR_Data(yChar&0x00ff);
LCD_WR_Data((yChar+15)>>8);
LCD_WR_Data((yChar+15)&0x00ff);
LCD_WR_REG(0x002c);
for(i=0;i<h;i++)
{
for(j=0; j<w; j++) LCD_RAM = *bitbmp++;
}
```



| NO | LOC.NO | DESCRIPTION | TYPE | SIZE | SPECIFICATION |
|---|---|---|---|---|---|
| 1 | U1 | LCD-CONTROL | SMD | LFCSP_WQ | SSD1963QL9 |
| 2 | U2 | IC | SMD | SSOP16 | ADS7846E |
| 3 | *U3* | *IC* | *SMD* | *SSOP16* | *TSC2003* |
| 4 | *U4* | *Regulator* | *SMD* | *SOT-25* | *AP7331-12WG-7* |
| 5 | U5 | Regulator | SMD | SOT-223 | AZ1117H-1.2TRE1 |
| 6 | *U6* | *IC* | *SMD* | *SSOP20* | *AR1020-I/SS* |
| 7 | U7,U8,U9 | DC CONVERTE | SMD | MSOP | CAT4238 |
| 8 | Y1 | X-TAL | SMD | ATS | 10.0MHz/12pF SX-1 |
| 9 | R1,R2,R8,R12,R15,R26,R27 | Resistor | SMD | 1608 | 4.7 KΩ |
| 10 | R3,R14 | Resistor | SMD | 1608 | 22 Ω |
| 11 | R4 | Resistor | SMD | 1608 | 100 KΩ |
| 12 | R5,R6 | Resistor | SMD | 1608 | 2.2 KΩ |
| 13 | R7,R9,R16,R18,R30,R31,R32,R33,R34,R35,R36,R37 | Resistor | SMD | 1608 | 0 Ω |
| 14 | R10,R17,R19 | Resistor | SMD | 1608 | NC |
| 15 | R11 | Resistor | SMD | 1608 | 470 Ω |
| 16 | R13 | Resistor | SMD | 1608 | 20 KΩ |
| 17 | R20,R21,R22 ,R23 ,R24,R25 | Resistor | SMD | 1608 | 10 Ω |
| 18 | R28,R29 | Resistor | SMD | 1608 | 10 KΩ |
| 19 | C1,C4 | Capacitor | SMD | 1608 | 5PF |
| 20 | C2,C5,C10,C13,C20,C21,C24,C31,C39 | Capacitor | SMD | A | 10uF/16V |
| 21 | C3,C6,C7,C9,C11,C12,C15,C17,C18,C19,C25,C26, C27,C28,C29,C30,C32,C33,C34,C35,C36,C37 | Capacitor | SMD | 1608 | 104 |
| 22 | C8,C14,C16 | Capacitor | SMD | B | 47uF/10V |
| 23 | C22,C23,C38 | Capacitor | SMD | B | 1uF/35V |
| 24 | D1 | DIODE | SMD | DO-214AC | 1N4007 |
| 25 | D2,D3 | LED | SMD | 1608 | GRN |
| 26 | D4,D5,D6 | DIODE | SMD | DO-214AC | SS14 |
| 27 | FB1,FB2,FB3,FB4 | BEAD | SMD | 2012 | 15 ohm |
| 28 | J1,J4,J12 | 커넥터 | SMD | 2.54p | HIF3BA-34PA-2.54DSA 또는=>34 핀 헤더 |
| 29 | J2,J3 | 커넥터 | SMD | 0.5mm 40 양면 | FPC TM-0520-0040 LMS700KF05 |
| 30 | J5 | 커넥터 | DIP | M5267-02 | CON2 |
| 31 | J9,J10 | 핀헤더 | DIP | 2x1 | CON2 |
| 32 | J6 | 커넥터 | SMD | FH12-6S-0.5SH(55) | CON8 |
| 33 | J7,J8,J11,J13 | 커넥터 | SMD | 52271-0490 | 터치(1개만 사용) |
| 34 | L1,L2,L3 | 코일 | SMD | 5034 | 22uH YC53RT-220M=>47uH |
| 35 | Q1 | TR | SMD | | MMBT3904(2N3904) |
| 36 | LMS700KF05 | LCD | DIP | | 7inch 640X480 |

### (7) 프로그램 (예)

① 핀 정의

```
#define LCD_DC                  GPIO_Pin_2 // GPIOB
#define LCD_CS                  GPIO_Pin_12// GPIOB
#define LCD_RD                  GPIO_Pin_13// GPIOB
#define LCD_WR                  GPIO_Pin_14// GPIOB
#define LCD_RST                 GPIO_Pin_15// GPIOB
#define LCD_DATA_GPIO           GPIOC


void SSD1963_WriteCommand(unsigned    int cmd)
{
        SSD1963_CS_LOW();
        SSD1963_DC_LOW();
        SSD1963_WR_LOW();
        SSD1963_DATA_GPIO(cmd);
        SSD1963_WR_HI();
        SSD1963_CS_HI();
}

void SSD1963_WriteData(unsigned int data)
{
        SSD1963_CS_LOW();
        SSD1963_DC_HI();
        SSD1963_WR_LOW();
        SSD1963_DATA_GPIO(data);
        SSD1963_WR_HI();
        SSD1963_CS_HI();
}
```

② SSD1963에 10MHz 크리스탈 사용시

```
SSD1963_WriteCommand(0x00E2); // Set the PLL (PLL Frequency = 10MHz*36/3 = 120MHz)
SSD1963_WriteData(0x0035);    //      M=35, 360Mhz = 10Mhz x (35+1)
SSD1963_WriteData(0x0002);      // N=2    (3-1),        120Mhz = 360Mhz / 3
SSD1963_WriteData(0x0054);//Dummy Byte
```

③ DOTCLK Frequence =24.5MHz 이므로

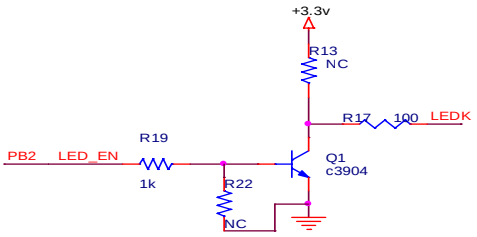
```
// Optimum Pixel Clock(PCLK) for SAMSUNG 7Inch LCD is 24.5Mhz,
// 24.5MHz = ( 120MHz * ( LCDC_FPR+ 1) ) / 2^20
// LCDC_FPR = 214083 = 0x034443
// Set LSHIFT(pixel clock) frequency.

SSD1963_WriteCommand(0x00E6);//LCD_PIXEL_CLOCK);//
SSD1963_WriteData(0x0003);//
SSD1963_WriteData(0x0044);//
SSD1963 _WriteData(0x0043);//
```

## 8) 백라이트

① 서브보드 적용

TFT LCD 화면을 비춰주는 것으로 반드시 전원을 인가 해야 TFT LCD 화면을 볼

수 있다. PB2 에 High 를 인가하면 R17에 Low가 되어 백라이트 전원이 들어온다. 프로그램 코딩은 LCD_BL_H; 일 때 화면이 나온다.




[그림 2.2.10] 백라이트 구동 회로

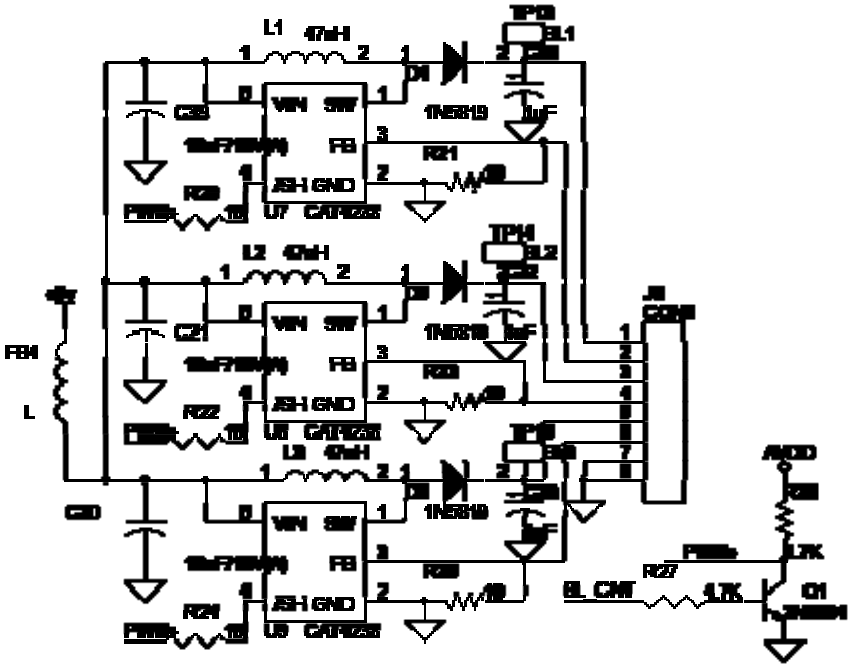
다른 회로로 사용할 때 R13,R22 저항이 필요할 수 있다. R13, R22는 연결 안 한다.

**LCD_BL_GPIO_Config(); 내용**

```
GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_AFIO, ENABLE);
GPIO_InitStructure.GPIO_Pin =LCD_BL;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

② SSD1963 적용

일반적으로 백라이트 전용 LED Drive IC 소자를 많이 사용하고 있다. 대표적으로 FAN5333, CAT4238TD, AAT3123, AAT3132, NCP5005 등 LCD 판넬에 LED 조건에 맞게 적용을 한다. 4.3 인치 정도는 LED Drive IC를 하나 정도 사용하지만 7인치 정도는 3개를 사용해야 하는 경우도 있다. 한 개로 사용할 경우 화면이 어둡거나, 열이 많이 발생한다. 또는 IC에서 LED 여러 개를 지원하는 소자를 선택하면 된다.



## 9) Touch

### 1. ADS7846

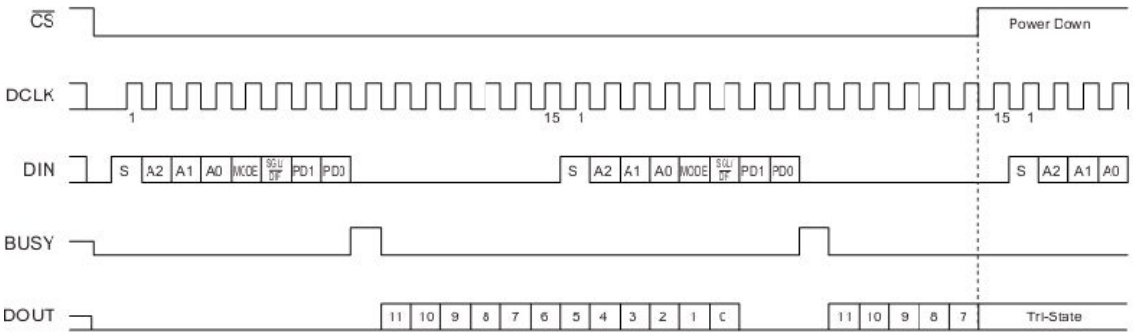
① **ADS7846**특 징

ADS7843이나 ADS7846 사용되는 핀 특성이 동일하다. .ADS7843/7846은 4선식 저항막 방식의 터치스크린 컨트롤 소자로 주요 특징으로는 내부 2.5V기준 전압을 이용하여 배터리 전압측정, 칩 온도측정과 터치 압력 측정기능이 있으며, SPI방식 통신으로 사용한다.

핀 구성

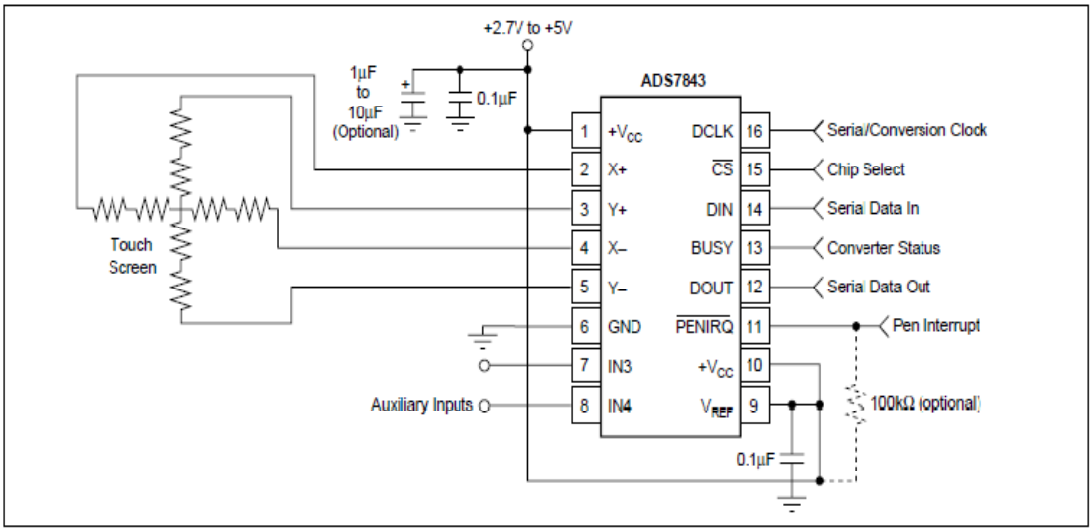
| 핀 번호 | 핀 이름 | 기능 요약 |
|---|---|---|
| 1 | +VCC | 전원 전압, 2.7V to 5V. |
| 2 | X+ | X+ 위치 입력 ADC1 |
| 3 | Y+ | Y+ 위치 입력 ADC2 |
| 4 | X- | X-위치 입력 |
| 5 | Y- | Y-위치 입력 |
| 6 | GND | 접지 |
| 7 | IN3 | AUX 입력 1. ADC3 (Battery Monitor Input => ADS7846) *1 |
| 8 | IN4 | AUX 입력 2, ADC4 (Auxiliary Input to ADC =>ADS7846) *2 |
| 9 | VREF | 기준 전압 |
| 10 | +VCC | 디지털 I/O 전원 전압, 2.7V to 5V. |
| 11 | /PENIRQ | PEN 인터럽트. 오픈 애노드 출력(저항 10kΩ~100kΩ 풀업저항 필요) |
| 12 | DOUT | 직렬 데이터 출력. 데이터는 DCLK 의 하강에지에 의해 이동.이 출력은 높은 임피던스일 때 /CS 가 High |
| 13 | BUSY | Busy 출력. 이 출력은 높은 임피던스일 때 /CS 가 High |
| 14 | DIN | 직렬 데이터 입력. /CS 는 Low, 데이터는 DCLK 의 상승에지에 의해 래치 |
| 15 | /CS | 칩 선택 입력. 변환 타이밍과 직렬 입/출력 레지스터 제어/CS = 1 파워 다운모드 ( A/D 컨버터 만) |
| 16 | DCLK | 확장 클럭 입력. 이 클럭은 SAR 변환 과정과 동기식 직렬 데이터 I/O 실행 |

ADS7843, ADS7846 차이점으로 7번 *1, 8번 *2 은 ADS7846의 특성을 나타낸다. 보드에서 이 부분을 사용하지 않기 때문에 구하기 쉬운 것으로 사용한다.




최대 변환율( 15Clocks-per-Conversion)

② 구성회로도




③ ADS7843/7846의 제어 명령

| Bit 7 (MSB) | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 (LSB) |
|---|---|---|---|---|---|---|---|
| S | A2 | A1 | A0 | MODE | SER/$\overline{DFR}$ | PD1 | PD0 |

ADS7843/7846의 핀

| 비트 | 이름 | 설명 |
|---|---|---|
| 7 | S | 시작 비트. 컨트롤 바이트는 DIN 에 의해 첫 번째 High 비트와 함께 시작. 새로운 컨트롤 바이트는 12 비트 변환모드에 15 클럭 주기 또는 8 비트 변환 모드에 11 클럭 주기로 시작. |
| 6~4 | A2 - A0 | 채널 선택 비트. SER//DFR 비트에 따라서 A/D MUX, 터치 드라이버 스위치, 기준 입력 설정 |
| 3 | MODE | 12/8 비트 변환 모드 선택 비트. 1 은 12 비트(LOW), 0 은 8 비트(HIGH). |
| 2 | SER/DFR | 기준 선택 비트. |
| | | A2-A0 비트에 따라서 A/D MUX, 터치 드라이버 스위치, 기준 입력 설정 |
| 1-0 | PD1 - PD0 | 파워-다운 모드 선택 비트. |

S는 스타트 비트로서 항상 1로 되며, MODE는 A/D 컨버터 모드 설정 비트이다.
M = 0 일때 12비트 분해능을 갖는 A/D 컨버터 모드 선택
M = 1 일때 8비트 분해능을 갖는 A/D 컨버터 모드 선택
여기서는 12 비트 분해능으로 1 을 사용 한다.

3비트로 구성된 A2-A0는 내부 A/D MUX로 A/D 변환한 값을 읽어올 때 사용하는 비트이다.
아래 표는 Single-Ended 기준 모드(SER/DFR HIGH).

| A2 | A1 | A0 | X+ | Y+ | IN3 | IN4 | –IN(1) | X SWITCHES | Y SWITCHES | +REF(1) | –REF(1) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | +IN | | | | GND | OFF | ON | $+V_{REF}$ | GND |
| 1 | 0 | 1 | | +IN | | | GND | ON | OFF | $+V_{REF}$ | GND |
| 0 | 1 | 0 | | | +IN | | GND | OFF | OFF | $+V_{REF}$ | GND |
| 1 | 1 | 0 | | | | +IN | GND | OFF | OFF | $+V_{REF}$ | GND |

NOTE: (1) Internal node, for clarification only—not directly accessible by the user.

TABLE I. Input Configuration, Single-Ended Reference Mode (SER/$\overline{DFR}$ HIGH).

아래 표는 Differential 기준 모드(SER/DFR LOW).

| A2 | A1 | A0 | X+ | Y+ | IN3 | IN4 | –IN(1) | X SWITCHES | Y SWITCHES | +REF(1) | –REF(1) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | +IN | | | | –Y | OFF | ON | +Y | –Y |
| 1 | 0 | 1 | | +IN | | | –X | ON | OFF | +X | –X |
| 0 | 1 | 0 | | | +IN | | GND | OFF | OFF | $+V_{REF}$ | GND |
| 1 | 1 | 0 | | | | +IN | GND | OFF | OFF | $+V_{REF}$ | GND |

NOTE: (1) Internal node, for clarification only—not directly accessible by the user.

TABLE II. Input Configuration, Differential Reference Mode (SER/$\overline{DFR}$ LOW).

SER/DFR PD1-PD0 은 단극성이나 차동 기준 입력 방식을 선택하는데 차동 기준 입력 방식을 채택한다. D1및 PD0 비트 는 0으로 사용하고 아래 그림 참고한다.

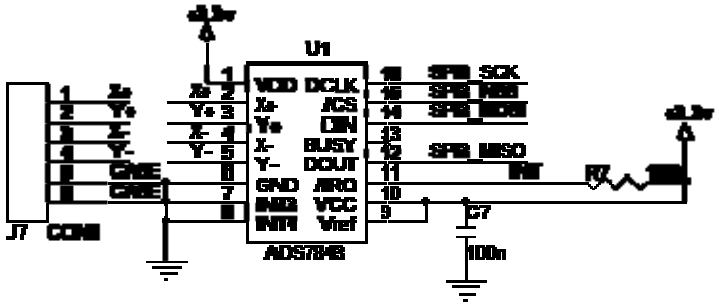
PD1및 PD0 비트 기능

| PD1 | PD0 | PENIRQ | DESCRIPTION |
|---|---|---|---|
| **0** | **0** | **사용** | **A/D 변환을 하지 않는 시간에는 파워 다운** |
| 0 | 1 | 금지 | 내부 기준전압은 OFF 상태, A/D 변환기는 ON 상태 |
| 1 | 0 | 사용 | 내부 기준전압은 ON 상태, A/D 변환기는 OFF 상태 |
| 1 | 1 | 금지 | 내부 기준전압과 A/D 변환기는 모두 ON 상태 |

④ STM32F103과 ADS7843/7846연결 배치
터치스크린 컨트롤에 필요한 X+, Y+, X-, Y-좌표 연결핀, SPI통신을 하기위한 DOUT, /CS, DCLK, DIN핀,/PENIRQ핀만을 사용하면 됩니다. /PENIRQ핀은 PEN 인터럽트 핀으로써, 4선식 저항막을 통한 터치스크린 인식시 L신호를 내보내어 다른 명령을 처리 중 이던 MCU에게 터치스크린이 눌렸음을 알려주는 인터럽트 핀이다.
터치스크린을 눌려졌을 때 변화하는 전압 값, 또한 A/D MUX채널을 변경하여 A/D변환 값을 측정하고, MCU와 SPI통신으로 해당 값을 읽는다.

| PB3 | PB4 | PB5 | PB6 | PB7 |
|---|---|---|---|---|
| SPI_CLK | SPI_MISO | SPI_MOSI | TP_CS | TP_INT |

SPI_CLK (STM32, PB3) =>DCLK (ADS7843/7846, 16핀)
SPI_MISO (STM32, PB4) =>DOUT (ADS784/78463, 12핀)
SPI_MOSI (STM32, PB5) =>DIN (ADS7843/7846, 14핀)
TP_CS (STM32, PB6) =>/CS (ADS7843/7846, 15핀)
TP_INT (STM32, PB7) =>/PENIRQ(ADS7843/7846, 11핀)

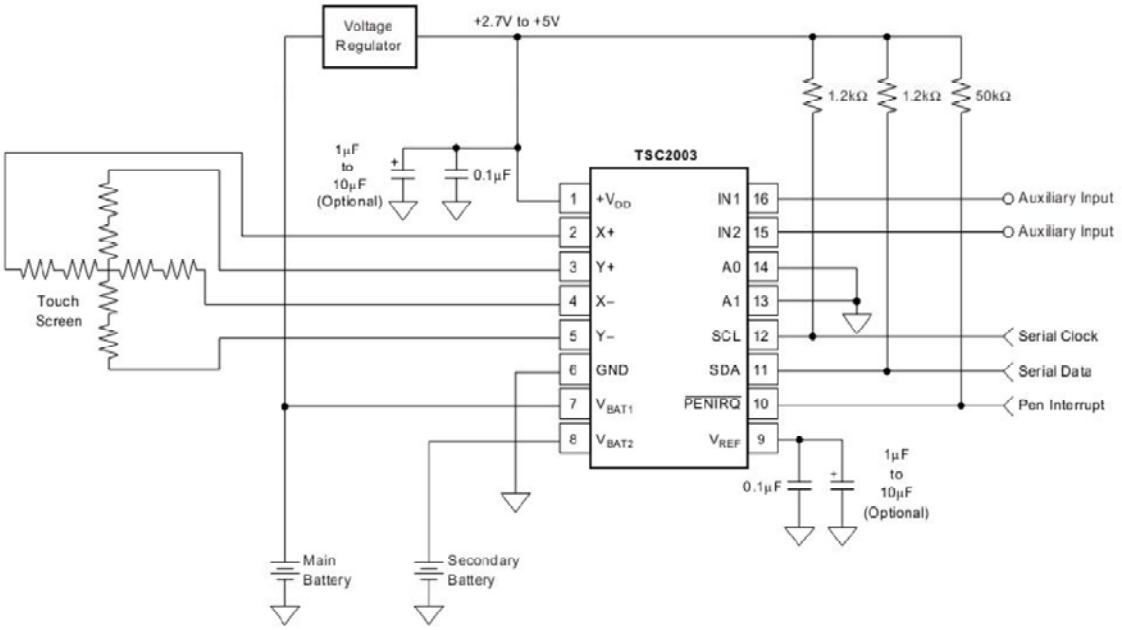



[그림 2.2.12] ADS7843/7846 회로도

2. **TSC2003**

| TSC DEVICE | ANALOG INTERFACE TYPE | DIGITAL INTERFACE TYPE | OPERATION SCHEME |
|---|---|---|---|
| ADS7843 | 4-wire | SPI | Normal Command-Based |
| ADS7845 | **5-wire** | SPI | Normal Command-Based |
| ADS7846 | 4-wire | SPI | Normal Command-Based |
| TSC2003 | 4-wire | **I²C** | Normal Command-Based |
| TSC2046 | 4-wire | SPI | Normal Command-Based |
| TSC2004 | 4-wire | **I²C** | Register-Based with Batch-Delay |
| TSC2005 | 4-wire | SPI | Register-Based with Batch-Delay |
| TSC2006 | 4-wire | SPI | Register-Based with Batch-Delay |
| TSC2007 | 4-wire | **I²C** | Advanced Command-Based |

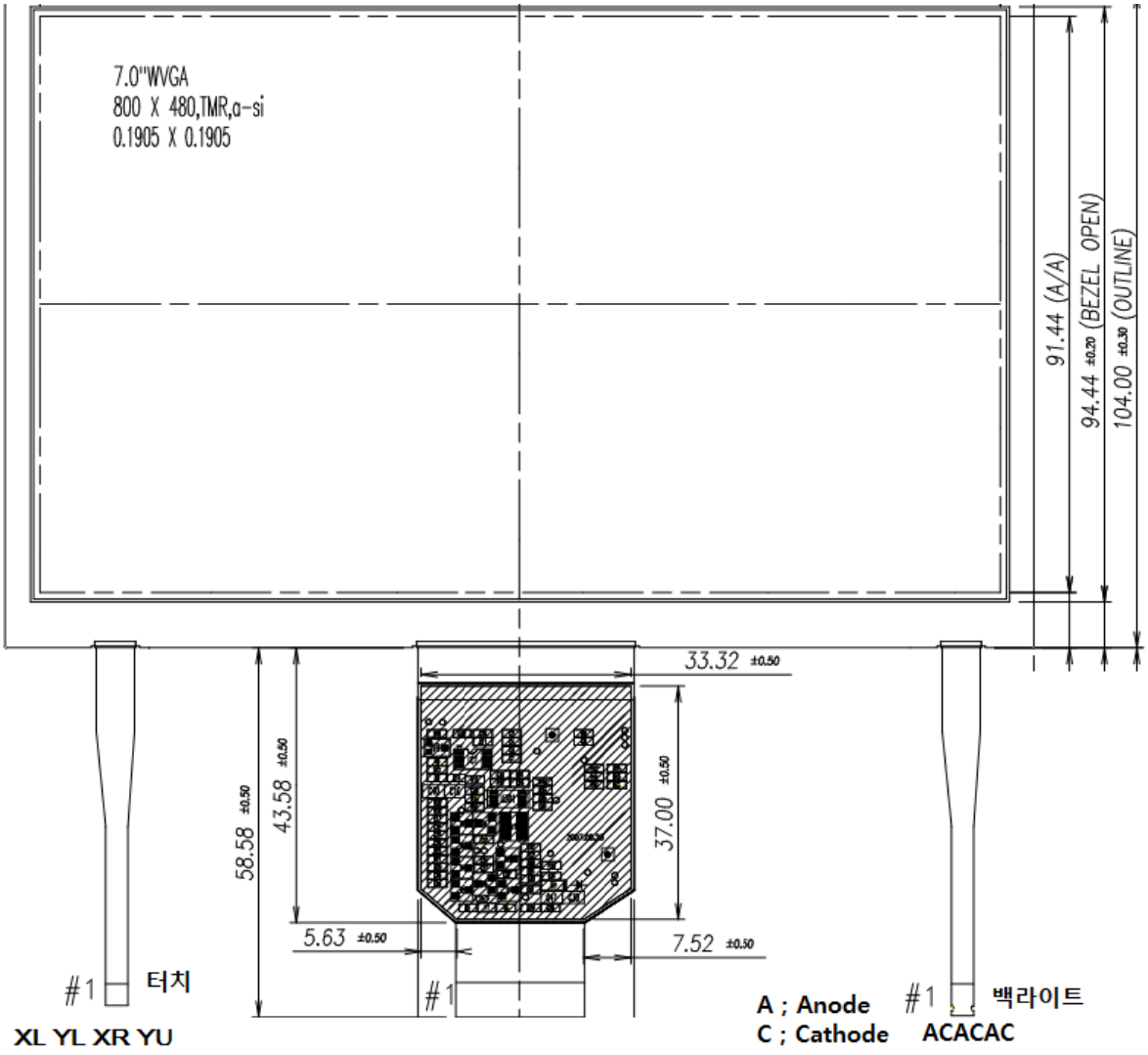
ADS7843/7846와 함께 많이 사용된다. 주요 특징으로는 외부 AUX입력, 내부 2.5V기준전압을 이용하여 배터리 2개의 전압측정, 온도측정과 터치 압력 측정기능이 있으며, I2C방식의 통신을 한다.




+$V_{DD}$ 전원핀과 GND, 터치스크린 컨트롤에 필요한 X+, Y+, X-, Y-좌표 연결핀, I2C통신을 하기위한 SDA, SCL핀, /PENIRQ핀만을 사용한다. /PENIRQ핀은 PEN 인터럽트 핀으로써, 4선식 저항막을 통한 터치스크린 인식시 L신호를 보내 다른 명령을 처리 중 이던 MCU에게 터치스크린이 눌렸음을 알려주는 인터럽트 핀이다.
전원 전압 2.5V ~ 5.25V까지의 동작 범위를 가지고 있고, I2C통신을 사용하며 동기식 직렬통신 방식으로 마스터가 원하는 시간에 모든 통신을 주도할 수 있고, 2선만으로 양방향 통신이 가능하여 널리 사용되는 통신 방식이다. 통신 속도는 Standard Mode(100kHz), Fast Mode(400kHz), High-Speed Mode(3.4MHz)를 지원한다.
앞서 특징에서 살펴본 배터리 전압 모니터, 칩 온도측정, 터치스크린 압력측정은 내부에 A/D MUX의 채널을 변경하여 원하는 해당 기능의 A/D 변환 값을 측정하며, 터치스크린을 눌려졌을 때 변화하는 전압 값 또한 A/D MUX채널을 변경하여 A/D변환 값을 측정하고 MCU와 I2C통신으로 해당 값을 읽는다.

LMS700KF06 외형 및 치수

| 터치 핀 번호 | 터치 기능 | ADS7846 핀 번호 | ADS7846 기능 |
|---|---|---|---|
| 1 | XL | 2 | X+ |
| 2 | YL | 3 | Y+ |
| 3 | XR | 4 | X− |
| 4 | YU | 5 | Y− |




### 3. Touch 커넥터

**1) I2812-7IPT2432A => PCB 직접 납땜, 터치 스크린 포함**

2) COM26T2844 => 터치 스크린 포함, 커넥터필요. 아래 [그림 2.2.11] 부품 필요

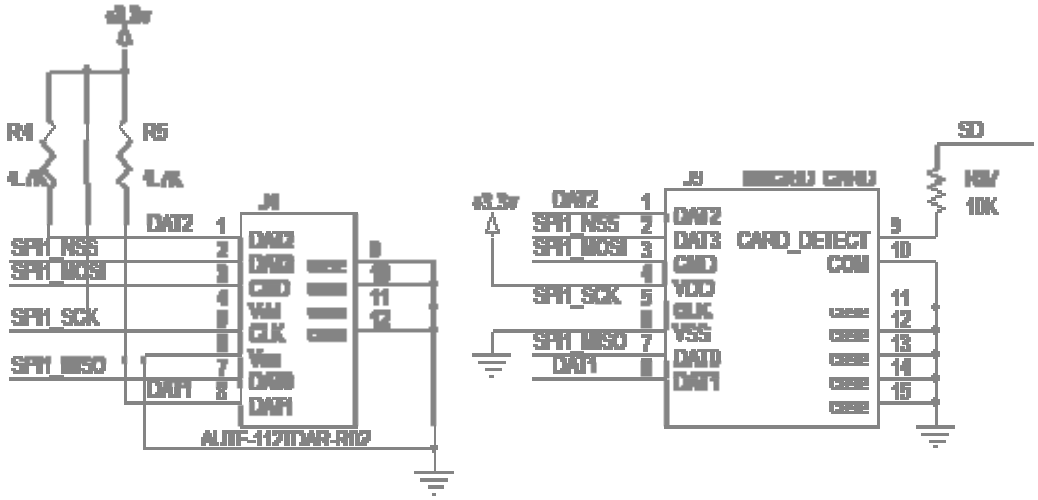

[그림 2.2.11] COM26T2844 에 필요한 커넥터

1) 52271-0490(Bottom) => 터치 스크린 커넥터(52207-0490 Top 사용안함)
2) NT-FPC0.3C-0.5-45P => PCB부착 커넥터와 LCD 커넥터 연결
3) NT-FFC-CON45P => PCB부착 커넥터

3. TG028HBZ43 => **PCB 직접 납땜, 터치 스크린 포함**

**10) SD Card (SPI 사용)**

SD Card 소켓은 [그림 2.2.13-2] 처럼 J4의 AUTF-112CTDAR-R02, J9의 T-Flash소켓(49225-0821) 둘 중 하나를 구하기 쉬운 것을 선택해 사용 할 수 있게 되어 있다.
추천 소켓은 J9의 T-Flash소켓(49225-0821) 으로 SD Card 넣고 빼기가 쉽게 되어 있다. AUTF-112CTDAR-R02는 밀고 소켓 윗부분을 올려야 열리게 되어 있다.
**여기서는 SD Memory Card(최대용량 2G), SDHC(4G) 프로그램으로 코딩으로 구분하여 프로그램이 작성 되어 있다.**




[그림 2.2.13-1] SD Card 회로도

[그림 2.2.13-2] SD Card 실물

**(1). SD 카드 개요**

SD 카드는 대용량 저장장치로 휴대용 기기에 널리 사용되고 있다. SD 카드에는 3 가지 전송모드(1-bit SD, 4-bit SD, SPI mode)가 있고 각 카드는 3 가지 전송방식을 모두 지원한다. (단 MicroSD 인 경우 SPI 모드는 옵션이다.) 일반적인 경우 25MHz 클럭까지 지원하고 High speed 인 경우 50MHz 클럭을 지원 한다. MMC 와도 대응하고 있으며 약간의 고려사항을 맞추면 상호 호환가능 하다.
SD 카드는 다른 플래시 메모리 컨트롤러와 달리 SPI 모드를 사용할 수 있어 일반 SPI 모듈을 사용하는 MCU 라면 큰 어려움 없이 접속하여 사용할 수 있다.

| 구분 | FAT12 | **FAT16(32)** | FAT32 |
|---|---|---|---|
| 사용 용도 | 플로피 | **저용량 HDD** | 고용량 HDD |
| 클러스터 표현 비트 수 | 12bit | **16bit** | 32bit(28bit 만 사용) |
| 최대 클러스터 개수 | 4,084 개 | **65,524 개** | 약 228 만개 |
| 최대 볼륨 크기 | 16MB | **2GB** | 2TB |
| 디렉토리당최대 파일 개수 | X | **65,535 개** | 65,535 개 |
| 루트디렉토리의 파일개수 제한 | 있음 | **있음** | 없음 |

**(2) SD 카드 특징**

여러가지 특징이 많은 가운데 이중 참고할 만한 부분은 먼저 **SD Memory Card 의 용량은 최대 2GB**, SDHC 는 2GB 이상 32GB 이고, SDXC 는 32GB 에서 2TB 이다. 규격은 구조적으로 차이가 있기 때문에 많이 알려지고 쉽게 제어할 수 있는 즉, 우리가 사용할 규격은 일반적은 SD 카드이나, SDHC 카드는 제어나 기타 사용에 널리 사용되고 있다. **여기서는 SD Memory Card 는 2G, SDHC 4G 로 테스트 하였다.**

**4. SD 카드의 실물**

[그림 2.2.14] SD Card 실물

**Secure Digital I/O Pinout**

| Pin # | SD 4-bit Mode | | SD 1-bit Mode | | SPI Mode | |
|---|---|---|---|---|---|---|
| 1 | CD/DAT[3] | Data Line 3 | N/C | Not Used | CS | Card Select |
| 2 | CMD | Command Line | CMD | Command Line | DI | Data Input |
| 3 | VSS1 | Ground | VSS1 | Ground | VSS1 | Ground |
| 4 | VDD | Supply Voltage | VDD | Supply Voltage | VDD | Supply Voltage |
| 5 | CLK | Clock | CLK | Clock | SCLK | Clock |
| 6 | Vss2 | Ground | Vss2 | Ground | Vss2 | Ground |
| 7 | DAT[0] | Data Line 0 | DATA | Data Line | DO | Data Output |
| 8 | DAT[1] | Data Line 1 / Interrupt | IRQ | Interrupt | IRQ | Interrupt |
| 9 | DAT[2] | Data Line 2 /Read Wait | RW | Read Wait | NC | Not Used |

| 커맨드 인덱스 | 인자 | 응답 | 데이터 전송 | 생략형 | 설 명 |
|---|---|---|---|---|---|
| CMD0 | None(0) | R1 | No | GO_IDLE_STATE | SD 모드에서 SPI 모드 변경 시 사용 |
| CMD1 | None(0) | R1 | No | SEND_OP_COND | 초기화 개시 |
| ACMD41(*1) | *2 | R1 | No | APP_SEND_OP_COND | SDC 전용. 초기화 시작 |
| CMD8 | *3 | R7 | No | SEND_IF_COND | SDC V2 전용. 동작 전압 확인 |
| CMD9 | None(0) | R1 | Yes | SEND_CSD | CSD 레지스터 읽기 |
| CMD10 | None(0) | R1 | Yes | SEND_CID | CID 레지스터 읽기 |
| CMD12 | None(0) | R1b | No | STOP_TRANSMISSION | 데이터 읽기 강제로 정지 |
| CMD16 | Blocklength[31:0] | R1 | No | SET_BLOCKLEN | 읽기 쓰기 블록 사이즈 변경 |
| CMD17 | Address[31:0] | R1 | Yes | READ_SINGLE_BLOCK | 싱글 블록 읽기 |
| CMD18 | Address[31:0] | R1 | Yes | READ_MULTIPLE_BLOCK | 멀티 블록 읽기 |
| CMD23 | Number ofblocks[15:0] | R1 | No | SET_BLOCK_COUNT | MMC 전용. 멀티 블록 읽기/쓰기. 커맨드로 전송 블록 수 설정 |
| ACMD23(*1) | Number ofblocks[22:0] | R1 | No | SET_WR_BLOCK_ERASE_COUNT | SDC 전용. 멀티 블록 쓰기 커맨드, 이전에 지울 블록 수 설정 |
| CMD24 | Address[31:0] | R1 | Yes | WRITE_BLOCK | 싱글 블록 쓰기 |
| CMD25 | Address[31:0] | R1 | Yes | WRITE_MULTIPLE_BLOCK | 멀티 블록 쓰기 |
| CMD55(*1) | None(0) | R1 | No | APP_CMD | 어플리케이션 특화 커맨드 |
| CMD58 | None(0) | R3 | No | READ_OCR | OCR 읽기 |

* 1 : ACMD<n>는 CMD55-CMD<n>의 커맨드 순서를 의미한다.
* 2 : 예약(0)[31], HCS[30], 예약(0)[29:0]
* 3 : 예약(0)[31:12], Supply Voltage(1)[11:8], Check Pattern(0xAA)[7:0]

<SPI 모드 초기화 순서>
MMC/SDC 서는 전원 ON 에 의해 우선 그것 본래의네이티부인 동작 모드가 됩니다. SPI 모드로 초기화 하려면 다음 순서를 밟을 필요가 있습니다.

전원 투입 또는 카드 삽입에 의해 카드는 250ms 이내, 최소 전압 2.0V 까지 전압이 상승해야한다. 전원이 안정 또는 카드 삽입을 검출하면 1ms 이상 기다리고, *DI (CMD 단자),CS 를 High 로 하고, SCLK 를 74clock 이상 (*400kHz 이하)*보내어 IDLE Satet* 에 코맨드 준비가 가능하다.
SPI 포트의 전송 clock·레이트를 100kHz～400kHz 로 설정했으면. CS 신호를 assert(Low)하고 *CMD0 명령을 보내면,* 소프트웨어·reset 를 한다. CMD0 는 CS 신호를 샘플 하고, Low 레벨의 경우는 SPI 모드로 들어 가는 중이다. CMD0 을 보낼 때는 SD 모드이고, 아직 SPI 모드는 아니기 때문에, 코맨드·패킷의 CRC 바이트 0x95 를 전송 해야 이 명령으로 SPI 모드로 들어 가면 R1 형태로 응답한다.
SD 카드가 V2.0 이상을 사용시 CMD8 을 보내 SD 카드 동작을 체크하고, 이때도 CRC 바이트 0x87 를 전송 해야 이 명령으로 SPI 모드로 들어 가면 R7 형태로 응답한다. 만약 CMD8 을 보내도 응답이 없으면 SD 카드는 V2.0 이하이다.
모드이고, 아직 SPI 모드는 아니기 때문에, 코맨드·패킷의 CRC 바이트 0x95 를 전송 해야 이 명령으로 SPI 모드로 들어 가면 R1 Response(0x01) 형태로 응답한다.

*CMD1* 를 보내면 카드는 초기화를 시작한다. 초기화의 종료를 polling 하기 때문에, CMD1 을 반복하 송신해 Response 를 조사합니다. 초기화가 종료하면, In Idle State 비트가 ( 0x00)Response 가 돌아간다. 카드에 의해서는 초기화로 *수백 ms 이상 걸린다* , 타임 아웃에는 여유를 들어 놓아야 한다. 이 처리가 종료한 시점부터

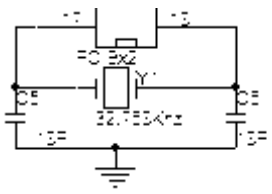
통상의 읽고 쓰기 동작이 가능하게 된다. SDC 그러면 CMD1 대신에 *ACMD41* 에서의 초기화가 권장 되고 있고 이를 사영한다. SD 카드와 MMC 양방에 대응하는 경우는, 먼저 ACMD41 을 보내어 봐 reject 되면 MMC 라고 판단해 CMD1 에 초기화를 행하는 것이 이상적이다.
CMD58 명령을 보내 CCS 정보를 읽고, 이 명령은 R3 형식으로 OCR 레지스터 값을 알려준다. CCS =0 이면 표준용량, CCS =1 이면 SDHC 카드이다. 이로서 SPI 초기화가 완료된다.

. CID 레지스터
CID 레지스터는 SD/MMC 카드의 고유한 정보를 담고 있다.
128 비트의 크기에 카드의 제품명, 시리얼번호 등의 내용을 가지고 있는데 그 내용은 아래와 같다.

| 이름 | 타잎 | 크기 (bit) | 위치 (bit) | 설명 |
|---|---|---|---|---|
| Manfacturer ID | Binary | 8 | 127~120 | 제조사의 고유 식별 번호 |
| OEM/Application ID | ASCII | 16 | 119~104 | OEM 혹은 카드 내용에 대한 식별번호 |
| Product Name (PNM) | ASCII | 40 | 103~64 | 제품명 |
| Product Revision (PRV) | BCD | 8 | 63~56 | 제품 리비젼. 2 자리의 BCD 번호 |
| Serial Number (PSN) | Binary | 32 | 55~24 | 시리얼 번호. 32 비트 정수 |
| Reserved | N/A | 4 | 23~20 | |
| Manufacture Date Code (MDT) | BCD | 12 | 19~8 | 제조일자 (yy-m 포맷) |
| CRC7 checksum (CRC) | Binary | 7 | 7~1 | CID 레지스터에 대한 CRC7 체크섬 |
| Reserved | N/A | 1 | 0 | 사용하지 않음. 항상 '1' |

**11) RTC**

실시간 클럭 (RTC, real-time clock)으로 시계나 달력을 만들 때 주로 이용하며마이컴에 일부분에 포함된 것을 이용하여 정확한 타이밍을 이용한다. 보통 32.768KHz를 사용한다. C5, C6은 N.C ~ 18P사이



[그림 2.2.13] RTC 회로도

# 제 3 장 STM32 실습

실습 위주이기 때문에 자세한 프로그램 관련한 설명은 생략한다. 필요한 부분에 대해서는 간단히 기술한다.

## 3.1 GPIO_LED

D1이 PB8 에 연결이 되어 있다. Low 일 때 LED 가 ON 되고, Low 일 때 High 이면 OFF된다.



[그림 3.1.1]회로

1_GPIO_LED 의 main 함수이다.

```
int main(void)
{               SystemInit();
                GPIO_Configuration();
      /* Infinite loop */
      while (1){          /*====D1-OFF=======*/
                          GPIO_SetBits(GPIOB , GPIO_Pin_8);
                          Delay(0xfffff);
                          Delay(0xfffff);
                          Delay(0x5ffff);
                          /*====D1-ON=======*/
                          GPIO_ResetBits(GPIOB , GPIO_Pin_8);
                          Delay(0xfffff);
                          Delay(0xfffff);
                          Delay(0x5ffff);
      }
}
```

Main 함수내의 GPIO_Configuration()는 PORTB 의 8번 핀을 출력으로 설정하는 방법이다.

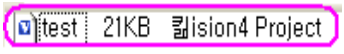
예1)

```
RCC->APB2ENR|=1<<3;      //PORT 클럭 활성화
GPIOB->CRH&=0XFFFFFFF0;
GPIOB->CRH|=0X00000003;//PB     8~15핀의 GPIO
GPIOB->ODR|=1<<8;          //PB8    GPIO Mode =>Output
```

예2)

```
   GPIO_InitTypeDef GPIO_InitStructure;
   RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOB , ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;                 // Port     PB8
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//    GPIO Speed 50MHz
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // GPIO Mode =>Output
   GPIO_Init(GPIOB, &GPIO_InitStructure);                      // Port B
```

여기서는 예1과 예2 가 같은 기능이나 이해하기가 쉬운 예2) 를 바탕으로 한다.

시뮬레이션을 해 보자. 프로그램은 구동은 1_GPIO_LED\test내의 아래 [그림 3.1.2]의 

더블클릭한다. 이것은 test.uvproj 에서 test 파일명이며,

파일 확장자는 uvproj 이며, 글씨가 이상한 글씨처럼 보이는 것이 μvision V4.0 을 나타낸다.
μvision 에서 확장자는 uvproj 가 구동 실행파일이다. test.hex 에서 확장자가 컴파일해서 생성된 파일로 이것을 다운로드 한다.

[그림 3.1.2] 1_GPIO_LED\test

다음은 에러없이 컴파일해서 시뮬레이션을 해보자. [그림 3.1.3] 왼쪽 Project 아래 Test 를 클릭하고 Option for Target 'Test'… 를 선택한다.

[그림 3.1.3] Option 선택

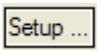
아래 [그림 3.1.4] 처럼 Use Simulator 가 체크되어 있어야 시뮬레이션을 할 수 있다.

[그림 3.1.4] Simulator 체크

그리고 아래 [그림 3.1.5]처럼 메뉴 바의 Debug 아래 툴을 선택하거나 [그림 3.1.6]의 그림 도구 바의  을 선택한다.

[그림 3.1.5] Debug 메뉴바

[그림 3.1.6] Debug 도구바

그림 도구 바의  Logic Analyzer 을 선택하고,  화면을 클릭하면 아래그림처럼 화면이 뜬다. [그림 3.1.7] 처럼  클릭하여 출력 포트 PORTB.8 을 쓰고 창을 닫는다.

[그림 3.1.7] 출력 포트설정

시뮬레이션 결과를 보기 위해  클릭 또는 Run(F5)을 한다. 파형을 전체 볼 때는  을 클릭하고, 리셋을 할 때  클릭하고, 중지할 때  클릭하면 된다. [그림 3.1.8]처럼 출력파형을 볼 수 있다.

[그림 3.1.8] 출력파형

그리고 또 다른 결과를 보는 방법은 아래 [그림 3.1.9] 처럼 들어간다.

[그림 3.1.9] 포트 설정

아래 그림[그림 3.1.9] 처럼 포트 PORTB.8 이 체크 상태를 반복하는 것을 알 수 있다.



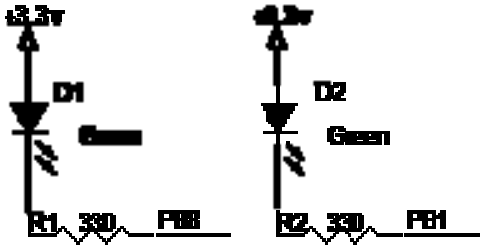
[그림 3.1.9] 실험 결과

**실행결과 보기**

1_GPIO_LED 테스트 결과이다. LED D1 이 깜빡이는 것을 알 수 있다.

[그림 3.1.10] 1_GPIO_LED 동작

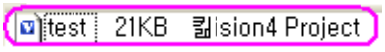
## 3.2 GPIO_LED




[그림 3.2.1] 2_GPIO_LED 회로도

LED 2개가 서로 교차하면서 깜빡이는 예제이다.

```
int main(void)
{               SystemInit();
                GPIO_Configuration();
while (1){ /* Infinite loop */
                        /* D1-ON D2-OFF */
                        GPIO_ResetBits(GPIOB , GPIO_Pin_8);          // D1 ON
                        GPIO_SetBits(GPIOB , GPIO_Pin_1);            // D2 OFF
                        Delay(0xfffff);
                        Delay(0xfffff);
                        Delay(0x5ffff);
                        /* D2-ON D1-OFF */
                        GPIO_ResetBits(GPIOB , GPIO_Pin_1);           // D2 ON
                        GPIO_SetBits(GPIOB , GPIO_Pin_8);             // D1 OFF
                        Delay(0xfffff);
                        Delay(0xfffff);
                        Delay(0x5ffff);
    }
}
```

프로그램은 구동은 2_GPIO_LED\test내의 

더블클릭한다. test.hex 에서 확장자가 컴파일해서 생성된 파일로 이것을 다운로드 한다. 시뮬레이션은 왼쪽 Project 아래 Test 를 클릭하고 Option for Target ‘Test’… 를 선택한다.

[그림 3.2.2] Option

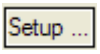
아래 그림처럼 Use Simulator 가 체크되어 있어야 시뮬레이션을 할 수 있다.

[그림 3.2.3] Use Simulator 체크

그리고 아래 그림처럼 메뉴 바의 Debug 아래 툴을 선택하거나, [그림 3.2.5] 도구 바의  을 선택한다.

[그림 3.2.4] Debug 메뉴 바

[그림 3.2.5] Debug 도구 바

그림 도구바의  Logic Analyzer 을 선택하고,  화면을 클릭하면 아래그림처럼 화면이 뜬다. [그림 3.2.6]  클릭하여 출력 포트 PORTB.1 을 쓴다.

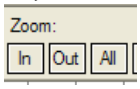
[그림 3.2.6] 포트 설정

그리고, 다시  클릭하여 출력 포트 PORTB.8 을 쓰고 Close 버튼을 클릭하고 창을 닫는다.

[그림 3.2.7] 출력 포트 설정

아래 그림처럼 PORTB.1, PORTB.8 창이 생성이 된다.

[그림 3.2.8] PORTB.1, PORTB.8 생성

 또는 F5 를 누르고 

파형의 TIME 을 셋팅하면 아래 [그림 3.2.9] 처럼 출력 파형결과를 얻을 수 있다.

[그림 3.2.9] 출력 파형결과

또 다른 방법은 이 상태에서 메뉴바의 Peripherals에서 General Purpose I/O 의 GPIOB 를 선택하면, [그림 3.2.1] 처럼 PORTB.1, PORTB.8 서로 교대하며 깜빡이는 알 수 있다.

[그림 3.2.10] 실행 결과

**실제 실행동작**

[그림 3.2.11] 동작

### 3.3 SysTick

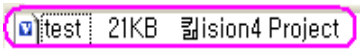
위의 예제 1_GPIO_LED 과 2_GPIO_LED 는 아래 함수들을 사용하였다.

```
void    Delay (uint32_t nCount)
{
  for(; nCount != 0; nCount--);
}
```

Delay 함수는 프로그램상 시간 지연을 목적으로 많이 사용하고 있다. CPU 에서 보면 단순한 숫자감소일 뿐, 그냥 프로그램 코드이다. 이유는 Delay 함수에서 사용된 변수, 매개 변수들은 Delay 함수 호출 시 Stack 에 생성됐다가 종료 후 Stack 에서 모두 사라지기 때문이다. 따라서 이는 많은 오차를 갖게 된다.
이를 해결하기 위해 SysTick 을 이용한 프로그램을 사용한다.

```
int main(void)
{
    GPIO_Configuration();
    delay_init();
    /* Infinite loop */
    while (1)
        {       //D2-OFF D1-ON
                GPIO_SetBits(GPIOB , GPIO_Pin_1);          //D2
                GPIO_ResetBits(GPIOB , GPIO_Pin_8);        //D1
                delay_ms(1000);     /* delay 1000ms */

                //D2-ON D1-OFF
                GPIO_ResetBits(GPIOB , GPIO_Pin_1);
                GPIO_SetBits(GPIOB , GPIO_Pin_8);
                delay_us(1000*1000);    /* delay 1000ms */
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////
void delay_init(void)
{
    RCC_ClocksTypeDef RCC_ClocksStatus;

    RCC_GetClocksFreq(&RCC_ClocksStatus);
    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
        SysTick_ITConfig(DISABLE);
    delay_fac_us = RCC_ClocksStatus.HCLK_Frequency / 8000000;
    delay_fac_ms = RCC_ClocksStatus.HCLK_Frequency / 8000;
}
void delay_us(u32 Nus)
{
    SysTick_SetReload(delay_fac_us * Nus);
    SysTick_CounterCmd(SysTick_Counter_Clear);
    SysTick_CounterCmd(SysTick_Counter_Enable);
    do
    {
        Status = SysTick_GetFlagStatus(SysTick_FLAG_COUNT);
    }while (Status != SET);
    SysTick_CounterCmd(SysTick_Counter_Disable);
        SysTick_CounterCmd(SysTick_Counter_Clear);
}
////////////////////////////////////////////////////////////////////////////////////////////////
```

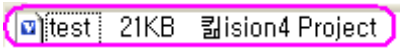
```
void delay_ms(uint16_t nms)
{
    uint32_t temp = delay_fac_ms * nms;

    if (temp > 0x00ffffff)
    {
        temp = 0x00ffffff;
    }
    SysTick_SetReload(temp);
    SysTick_CounterCmd(SysTick_Counter_Clear);
    SysTick_CounterCmd(SysTick_Counter_Enable);
    do
    {
        Status = SysTick_GetFlagStatus(SysTick_FLAG_COUNT);
    }while (Status != SET);
    SysTick_CounterCmd(SysTick_Counter_Disable);
          SysTick_CounterCmd(SysTick_Counter_Clear);
    }
}
```

프로그램은 구동은 3_SysTick\test내의 

더블클릭한다.
test.hex 에서 확장자가 컴파일해서 생성된 파일로 이것을 다운로드 한다.
2_GPIO_LED 같은 결과이지만 원하는 시간에 대해서는 정확하다.

## 3.4 USART

메인보드 S2스위치를 EX(실행)으로 위치해 놓고 통신 프로그램을 실행한다. .실행 뒤에 [그림 3.4.6] 처럼 환경 설정을 한다.

프로그램은 구동은 4_USART\test내의 

더블클릭한다. test.hex 에서 확장자가 컴파일해서 생성된 파일로 이것을 다운로드 한다.

통신프로그램 실행할 때는 

다운로드 프로그램을 실행하면 안된다. 같은 포트를 사용하기 때문에 충돌하기 때문이다. 아래 그림처럼 포트 설정을 한다.




[그림 3.4.6] 환경 설정

6. 4_USART main 함수 프로그램

```
int main(void)
{               SystemInit();
                GPIO_Configuration();
                USART_Configuration();
                delay_init();
                printf("\r\n");
                printf("*       STM32F103    Board   ! ^_^    *\r\n");
printf("*    USART Test,    BaudRate = 115200    *\r\n");
      while (1)
                {           GPIO_ResetBits(GPIOB , GPIO_Pin_8);
                              printf("*    GPIOB8 D1 ON      *\r\n");
                              delay_us(1000*1000);    /* delay 1000ms */
                            GPIO_SetBits(GPIOB , GPIO_Pin_8);
                              printf("*    GPIOB8 D1 OFF     *\r\n");
                            delay_ms(1000);      /* delay 1000ms */
                }
}
```

7. 통신 시뮬레이션

이번에는 USART 통신 프로그램을 시리얼 포트로 문자 전송하는 것을 시뮬레이션

하는 과정이다.

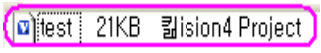
[그림 3.4.7] 상단 오른쪽은 1. 디버그 시작과 끝을 할 때 사용하고, 2. UART 선택한다. 3. 시뮬레이션 결과를 실행 하기 위해 클릭하고, 4. 프로그램을 작성한 것을 출력 해 볼 수 있다. 5. 리셋을 클릭해 다시 실행된 결과를 볼 수 있다. 지금까지 테스트 해 본 결과 하드 디버깅도 편하고 장비에 다운로드 안하고도 미리 알 수 있어 개발에 편리하다.

[그림 3.4.7] 통신 시뮬레이션

## 3.5 ADC_USART


회로도 가변저항 실물

[그림 3.5.1] ADC_USART

프로그램은 구동은 5_ADC_USART\test내의 

더블클릭한다.
test.hex 에서 확장자가 컴파일해서 생성된 파일로 이것을 다운로드 한다.

통신프로그램 실행할 때는 

다운로드 프로그램을 실행하면 안된다. 같은 포트를 사용하기 때문에 충돌하기 때문이다. 아래 그림처럼 포트 설정을 한다.
가변 저항을 돌리면 전압값이 바뀌는 것을 알 수 있다.

[그림 3.5.2] ADC_USART 실험결과

## 3.6 TFT_LCD

프로그램은 구동은 6_TFT_LCD\test내의 더블클릭한다. Test.hex 에서 확장자가 컴파일해서 생성된 파일로 다운로드 한다.

### A. TFT LCD 초기화

8비트 / 16비트 무관하게 공통으로 사용할 수 있게 되어 있으나, 8비트로 되어 있다. 초기화는 아래 I2812-7IPT2432A 의 void LCD_Init(void) 함수처럼 8비트 16비트 무관하다.

#### 1. I2812-7IPT2432A 초기화

```
void LCD_Init(void)
{
    delay_ms(1000);
    LCD_BL_GPIO_Config();
    LCD_GPIO_OUTPUT();
    LCD_Set_GPIO_Config();
    LCD_BL_H;
    LCD_CS_H;      LCD_RS_H;      LCD_RD_H;      LCD_WR_H;
    LCD_RST_H;     delay_ms(10);  LCD_RST_L;     delay_ms(50);
    LCD_RST_H;     delay_ms(100);
        //initializing funciton 1
    LCD_WR_REG(0x00E7,0x0010);
     LCD_WR_REG(0x0000,0x0001);
     LCD_WR_REG(0x0001,0x0100); // set SS and SM bit
     LCD_WR_REG(0x0002,0x0700); // set 1 line inversion
       // LCD_WR_REG(0x03, 0x10B0); // set GRAM write direction and BGR=1.
#if   ID_AM==000
       LCD_WR_REG(0x0003,0x1000);// TFM=0,TRI=0,SWAP=1,16 bits system interface   swap RGB to BRG,
 #elif ID_AM==001
                LCD_WR_REG(0x0003,0x1008);
#elif ID_AM==010
                LCD_WR_REG(0x0003,0x1010);
#elif ID_AM==011
                LCD_WR_REG(0x0003,0x1018);
#elif ID_AM==100
                LCD_WR_REG(0x0003,0x1020);
#elif ID_AM==101
                LCD_WR_REG(0x0003,0x1028);
#elif ID_AM==110
                LCD_WR_REG(0x0003,0x1030);
#elif ID_AM==111
                LCD_WR_REG(0x0003,0x1038);
#endif
        LCD_WR_REG(0x0004, 0x0000); // Resize register
        LCD_WR_REG(0x0008, 0x0207); // set the back porch and front porch
        LCD_WR_REG(0x0009, 0x0000); // set non-display area refresh cycle ISC[3:0]
        LCD_WR_REG(0x000A, 0x0000); // FMARK function
        LCD_WR_REG(0x000C, 0x0001); // RGB interface setting
        LCD_WR_REG(0x000D, 0x0000); // Frame marker Position
        LCD_WR_REG(0x000F, 0x0000); // RGB interface polarity
//Power On sequence //
        LCD_WR_REG(0x0010, 0x0000); // SAP, BT[3:0], AP, DSTB, SLP, STB
```

```
            LCD_WR_REG(0x0011, 0x0007); // DC1[2:0], DC0[2:0], VC[2:0]
            LCD_WR_REG(0x0012, 0x0000); // VREG1OUT voltage
            LCD_WR_REG(0x0013, 0x0000); // VDV[4:0] for VCOM amplitude
               LCD_WR_REG(0x0007, 0x0001); //

            delay_ms(2000); // Dis-charge capacitor power voltage
            LCD_WR_REG(0x0010, 0x1590); // SAP, BT[3:0], AP, DSTB, SLP, STB
            LCD_WR_REG(0x0011, 0x0227); // DC1[2:0], DC0[2:0], VC[2:0]
            delay_ms(1000); // Delay 50ms

            LCD_WR_REG(0x0012, 0x009c); // Internal reference voltage= Vci;
            delay_ms(1000); // Delay 50ms
            LCD_WR_REG(0x0013, 0x1900); // Set VDV[4:0] for VCOM amplitude
            LCD_WR_REG(0x0029, 0x0023); // Set VCM[5:0] for VCOMH
            LCD_WR_REG(0x002B, 0x000E); // Set Frame Rate
            delay_ms(500); // Delay 50ms

#if     ID_AM==000
                        LCD_WR_REG(0x0020,0x00ef);//GRAM
                        LCD_WR_REG(0x0021,0x013f);
#elif ID_AM==001
                        LCD_WR_REG(0x0020,0x00ef);
                        LCD_WR_REG(0x0021,0x013f);
#elif ID_AM==010
                        LCD_WR_REG(0x0020,0x0000);
                        LCD_WR_REG(0x0021,0x013f);
#elif ID_AM==011
                        LCD_WR_REG(0x0020,0x0000);
                        LCD_WR_REG(0x0021,0x013f);
#elif ID_AM==100
                        LCD_WR_REG(0x0020,0x00ef);
                        LCD_WR_REG(0x0021,0x0000);
#elif ID_AM==101
                        LCD_WR_REG(0x0020,0x00ef);
                        LCD_WR_REG(0x0021,0x0000);
#elif ID_AM==110
                        LCD_WR_REG(0x0020,0x0000);
                        LCD_WR_REG(0x0021,0x0000);
#elif ID_AM==111
                        LCD_WR_REG(0x0020,0x0000);
                        LCD_WR_REG(0x0021,0x0000);
#endif
// ---------- Adjust the Gamma Curve ----------//
                LCD_WR_REG(0x0030,0x0000);
                LCD_WR_REG(0x0031,0x0305);
                LCD_WR_REG(0x0032,0x0003);
                LCD_WR_REG(0x0035,0x0304);
                LCD_WR_REG(0x0036,0x000f);
                LCD_WR_REG(0x0037,0x0407);
                LCD_WR_REG(0x0038,0x0204);
                LCD_WR_REG(0x0039,0x0707);
                LCD_WR_REG(0x003C,0x0403);
                LCD_WR_REG(0x003D,0x1604);
```

```
//------------------ Set GRAM area ---------------//
            LCD_WR_REG(0x0050, 0x0000); // Horizontal GRAM Start Address
            LCD_WR_REG(0x0051, 0x00EF); // Horizontal GRAM End Address
            LCD_WR_REG(0x0052, 0x0000); // Vertical GRAM Start Address
            LCD_WR_REG(0x0053, 0x013F); // Vertical GRAM Start Address
            LCD_WR_REG(0x0060, 0xA700); // Gate Scan Line
            LCD_WR_REG(0x0061, 0x0001); // NDL,VLE, REV
            LCD_WR_REG(0x006A, 0x0000); // set scrolling line
//-------------- Partial Display Control ---------//
            LCD_WR_REG(0x0080, 0x0000);
            LCD_WR_REG(0x0081, 0x0000);
            LCD_WR_REG(0x0082, 0x0000);
            LCD_WR_REG(0x0083, 0x0000);
            LCD_WR_REG(0x0084, 0x0000);
            LCD_WR_REG(0x0085, 0x0000);
//-------------- Panel Control -------------------//
            LCD_WR_REG(0x0090, 0x0010);
            LCD_WR_REG(0x0092, 0x0600);
            LCD_WR_REG(0x0093, 0x0003);
            LCD_WR_REG(0x0095, 0x0110);
            LCD_WR_REG(0x0097, 0x0000);
            LCD_WR_REG(0x0098, 0x0000);
            LCD_WR_REG(0x0007, 0x0133); // 262K color and display ON

            LCD_WR_REG(0x0020, 0x0000);// 262K color and display ON
            LCD_WR_REG(0x0021, 0x0000);// 262K color and display ON

              LCD_Clear(0xffff); // white
              delay_ms(2000); // Delay 5ms
}
```

## 2. COM26T2844 초기화

```
void Lcd_Initialize(void)
{
            u16 i;
            Delay_nms(10);
            Lcd_Light_ON;
// Before power supply startup setting
LCD_WR_REG(0x0111,0x1000);
LCD_WR_REG(0x0110,0x0100);
LCD_WR_REG(0x0111,0x000c);
LCD_WR_REG(0x0110,0x0101);
// delay_ms(1);

LCD_WR_REG(0x0111,0x1000);
LCD_WR_REG(0x0110,0x0101);
// delay_ms(1);
LCD_WR_REG(0x0111,0x1202);
LCD_WR_REG(0x0110,0x0102);
// delay_ms(1);

// Power supply startup(1) setting
```

```
LCD_WR_REG(0x0100,0x4010);

LCD_WR_REG(0x0111,0x00bc);
LCD_WR_REG(0x0110,0x0100);
LCD_WR_REG(0x0111,0x1200);
LCD_WR_REG(0x0110,0x0100);
LCD_WR_REG(0x0111,0x001c);
LCD_WR_REG(0x0110,0x0101);
// Power supply startup(2) setting
LCD_WR_REG(0x0111,0x1a00);
LCD_WR_REG(0x0110,0x0100);
// Other mode sett
LCD_WR_REG(0x0111,0x0200);
LCD_WR_REG(0x0110,0x0103);
LCD_WR_REG(0x0111,0x1506);
LCD_WR_REG(0x0110,0x0103);
LCD_WR_REG(0x0111,0x0c41);
LCD_WR_REG(0x0110,0x0104);
// Display control register setting *** datasheet 16 page
LCD_WR_REG(0x0001,0x0000);     //ss=0        mirror 0100
LCD_WR_REG(0x0002,0x0700);
LCD_WR_REG(0x0003,0x0030);     //BGR =0      HWM=0, ID1:ID0 =11, AM=0
LCD_WR_REG(0x0008,0x0506);    //FP3:FP0=5, BP2:BP0=6
LCD_WR_REG(0x0009,0x0001);
LCD_WR_REG(0x000b,0x0000);    //FRCON=0, COL1:0=00 (262144色)
LCD_WR_REG(0x000d,0x0010);
LCD_WR_REG(0x0012,0x0000);
LCD_WR_REG(0x0013,0x0002);
LCD_WR_REG(0x0015,0x0000);
LCD_WR_REG(0x001b,0x0010);
LCD_WR_REG(0x001c,0x0000);

// Tset register setting
  LCD_WR_REG(0x0205,0x0000);
  LCD_WR_REG(0x070a,0x7800);
  LCD_WR_REG(0x070b,0x0000);
  LCD_WR_REG(0x070c,0x0030);
  LCD_WR_REG(0x070d,0x0000);
  LCD_WR_REG(0x070e,0x0000);
  LCD_WR_REG(0x070f,0x2100);
  LCD_WR_REG(0x0710,0x0000);

// Address setting
LCD_WR_REG(0x0400,0x0031);
LCD_WR_REG(0x0401,0x0000);
LCD_WR_REG(0x0402,0x0000);
LCD_WR_REG(0x0403,0x018f);
LCD_WR_REG(0x0404,0x0000);

// Y register setting
LCD_WR_REG(0x0300,0x0503);
LCD_WR_REG(0x0301,0x0403);
LCD_WR_REG(0x0302,0x0404);
LCD_WR_REG(0x0303,0x0303);
```

```
LCD_WR_REG(0x0304,0x0302);
LCD_WR_REG(0x0305,0x0203);
LCD_WR_REG(0x0306,0x0e14);
LCD_WR_REG(0x0307,0x0503);
LCD_WR_REG(0x0308,0x0403);
LCD_WR_REG(0x0309,0x0404);
LCD_WR_REG(0x030a,0x0303);
LCD_WR_REG(0x030b,0x0302);
LCD_WR_REG(0x030c,0x0203);
LCD_WR_REG(0x030d,0x1f10);

// Memory access
LCD_WR_REG(0x0210,0x0000);// Horizontal Start Address ( y = 0    )
LCD_WR_REG(0x0211,0x00ef);// Horizontal End Address      ( y = 239)
LCD_WR_REG(0x0212,0x0000);// Vertical Start Address      ( x = 0    )
LCD_WR_REG(0x0213,0x018f);// Vertical End Address         ( x = 339)

//##### Display On Sequence #####//
LCD_WR_REG(0x0100,0x4110);
LCD_WR_REG(0x0111,0x061c);
LCD_WR_REG(0x0110,0x0101);
LCD_WR_REG(0x0007,0x0001);
//delay_ms(45);
LCD_WR_REG(0x0111,0x161b);
LCD_WR_REG(0x0110,0x0101);
LCD_WR_REG(0x0007,0x0003);
//delay_ms(1);

LCD_WR_REG(0x0007,0x0113);
```

## 3.HX8347G 초기화

```
LCD_WR_REG(0x002E, 0x0088);                 // GDOFF
LCD_WR_REG(0x0029, 0x008F);                 // RTN
LCD_WR_REG(0x002B, 0x0004);                 // DUM
LCD_WR_REG(0x00E4, 0x0010);                 // EQ_S1
LCD_WR_REG(0x00E5, 0x0010);                 // EQ_S2
LCD_WR_REG(0x00E6, 0x0010);                 // EQ_S3
LCD_WR_REG(0x00E7, 0x0010);                 // EQ_S4
LCD_WR_REG(0x00EA, 0x0000);                 // STBA[15:8]
LCD_WR_REG(0x00EB, 0x0000);                 // STBA[7:0]
LCD_WR_REG(0x00EC, 0x003C);                 // PTBA[15:8]
LCD_WR_REG(0x00ED, 0x00C7);                 // PTBA[7:0]
LCD_WR_REG(0x00E8, 0x0070);                 // OPON[7:0]

LCD_WR_REG(0x0040, 0x0001);                 // Gamma 2.2 Setting
LCD_WR_REG(0x0041, 0x0007);
LCD_WR_REG(0x0042, 0x0007);
LCD_WR_REG(0x0043, 0x001E);
LCD_WR_REG(0x0044, 0x001C);
LCD_WR_REG(0x0045, 0x003E);
LCD_WR_REG(0x0046, 0x001B);
LCD_WR_REG(0x0047, 0x006B);
LCD_WR_REG(0x0048, 0x0007);
```

```
LCD_WR_REG(0x0049, 0x0013);
LCD_WR_REG(0x004A, 0x001A);
LCD_WR_REG(0x004B, 0x0019);
LCD_WR_REG(0x004C, 0x0016);
LCD_WR_REG(0x0050, 0x0001);
LCD_WR_REG(0x0051, 0x0023);
LCD_WR_REG(0x0052, 0x0021);
LCD_WR_REG(0x0053, 0x0038);
LCD_WR_REG(0x0054, 0x0038);
LCD_WR_REG(0x0055, 0x003E);
LCD_WR_REG(0x0056, 0x0014);
LCD_WR_REG(0x0057, 0x0064);
LCD_WR_REG(0x0058, 0x0009);
LCD_WR_REG(0x0059, 0x0006);
LCD_WR_REG(0x005A, 0x0005);
LCD_WR_REG(0x005B, 0x0006);
LCD_WR_REG(0x005C, 0x0018);
LCD_WR_REG(0x005D, 0x00CC);

LCD_WR_REG(0x001B, 0x001A);                    // VRH=4.60V
LCD_WR_REG(0x001A, 0x0001);
LCD_WR_REG(0x0024, 0x0070);                    // VMH
LCD_WR_REG(0x0025, 0x0058);                    // VML
LCD_WR_REG(0x0023, 0x0078);                    // VMF

LCD_WR_REG(0x0018, 0x0036);                    // Power on Setting
LCD_WR_REG(0x0019, 0x0001);
LCD_WR_REG(0x0001, 0x0000);
LCD_WR_REG(0x001F, 0x0088);
Delay_nms(5);
LCD_WR_REG(0x001F, 0x0080);
Delay_nms(5);
LCD_WR_REG(0x001F, 0x0090);
Delay_nms(5);
LCD_WR_REG(0x001F, 0x00D4);
Delay_nms(5);
LCD_WR_REG(0x0017, 0x0005);                    // 16-bit/pixel = 65536 colors

LCD_WR_REG(0x0036, 0x0008);                    // SET PANEL (BGR_PANEL = 0)
LCD_WR_REG(0x0028, 0x0038);                    // Display ON Setting
Delay_nms(40);
LCD_WR_REG(0x0028, 0x003C);

LCD_WR_REG(0x0002, 0x0000); //write_data(0x0000,0x0000);
LCD_WR_REG(0x0003, 0x0000); //write_data(0x0000,0x0000); //Column Start
LCD_WR_REG(0x0004, 0x0000); //write_data(0x0000,0x0000);
LCD_WR_REG(0x0005, 0x00EF); //write_data(0x0000,0x00EF); //Column End
LCD_WR_REG(0x0006, 0x0000); //write_data(0x0000,0x0000);
LCD_WR_REG(0x0007, 0x0000); //write_data(0x0000,0x0000); //Row Start
LCD_WR_REG(0x0008, 0x0001); //write_data(0x0000,0x0001);
LCD_WR_REG(0x0009, 0x003F); //write_data(0x0000,0x003F); //Row End
```

①I I2812-7IPT2432A ②COM26T2844 ③**HX8347G** 공통으로 8비트, 16 비트일 때와 포트 상위,하위 사용할 때 다르므로 아래 프로그램을 참고 한다.

### ① 8bit TFT LCD 1

포트 Px0~7 (x 는 PA, PB, PB, PC 중 하나)를 사용할 때 이다. 아래 예제는 PC.
GPIO_Config 는 아래와 같이 코딩한다.

```
GPIO_InitStructure.GPIO_Pin =
GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7;

void LCD_WR_REG16(unsigned int index)
{
  LCD_CS_L;    LCD_RS_L;    LCD_RD_H;    LCD_WR_H;
  GPIOC->ODR= (GPIOC->ODR&0XFF00)| (unsigned char)(index>>8);    //   0~7
  LCD_WR_L;          LCD_WR_H;
  GPIOC->ODR=(GPIOC->ODR&0XFF00)|(unsigned char)index;            //   0~7
  LCD_WR_L;    LCD_WR_H;       LCD_CS_H;
}
void LCD_WR_DATA16(unsigned int data)
{
  LCD_CS_L;   LCD_RS_H;   LCD_RD_H;      LCD_WR_H;
  GPIOC->ODR=(GPIOC->ODR&0XFF00)| (unsigned char)(data>>8);      //   0~7
  LCD_WR_L;    LCD_WR_H;
  GPIOC->ODR= (GPIOC->ODR&0XFF00)|(unsigned char)data;            //   0~7
  LCD_WR_L;    LCD_WR_H;    LCD_CS_H;
}
```

### ② 8bit TFT LCD 2

포트 Px8~15 (x 는 PA, PB, PB, PC 중 하나)를 사용 할 때 이다. 아래 예제는 PC.
GPIO_Config 는 아래와 같이 코딩한다.

```
GPIO_InitStructure.GPIO_Pin                                                                                    =
GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;

void LCD_WR_REG16(unsigned int index)
{
  LCD_CS_L;    LCD_RS_L;    LCD_RD_H;    LCD_WR_H;
  GPIOC->ODR=((GPIOC->ODR&0x00ff)|(index&0xff00));   //          8~15
  LCD_WR_L;          LCD_WR_H;      delay_us(1);
  GPIOC->ODR=((GPIOC->ODR&0x00ff)|(index<<8));       //          8~15
  LCD_WR_L;    LCD_WR_H;       LCD_CS_H;
}


void LCD_WR_DATA16(unsigned int data)
{
  LCD_CS_L;   LCD_RS_H;   LCD_RD_H;      LCD_WR_H;
  GPIOC->ODR=((GPIOC->ODR&0x00ff)|(data&0xff00)); //   8~15
  LCD_WR_L;    LCD_WR_H;
  GPIOC->ODR=((GPIOC->ODR&0x00ff)|(data<<8));            //   8~15
  LCD_WR_L;    LCD_WR_H;    LCD_CS_H;
}
```

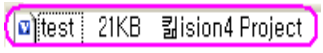
### ③ 16bit TFT LCD

포트 Px0~15 (x 는 PA, PB, PB, PC 중 하나)를 사용 할 때 이다.
GPIO_Config 는 아래와 같이 코딩한다.

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;

void LCD_WR_REG16(unsigned int index)
```

```
{
LCD_CS_L;     LCD_RS_L;     LCD_RD_H;     LCD_WR_H;
DATA_LCD_PORT=index;                                    //0~15 16bit
LCD_WR_L;     LCD_WR_H;    LCD_CS_H;
}
void LCD_WR_DATA16(unsigned int data)
{
 LCD_CS_L;    LCD_RS_H;    LCD_RD_H;       LCD_WR_H;
 DATA_LCD_PORT=data;                                    //0~15 16bit
 LCD_WR_L;     LCD_WR_H; LCD_CS_H;
```

### 3.7 TFT_ADS7843

프로그램은 구동은 7_TFT_ads7843\test내의 

더블클릭한다.
test.hex 에서 확장자가 컴파일해서 생성된 파일로 이것을 다운로드 한다.

[그림 3.7.1] TFT_ADS7843 테스트 화면

통신프로그램 실행할 때는 

다운로드 프로그램을 실행하면 안된다. 같은 포트를 사용하기 때문에 충돌하기 때문이다. 아래 그림처럼 포트 설정을 한다.

TFT_ADS7843에 사용되는 ADS7843이나 ADS7846 사용되는 핀 특성이 동일하다.

## 3.8 TFT_OV7670

프로그램은 구동은 8_TFT_OV7670\TFT_LCD.uvproj 더블클릭한다.
8_TFT_OV7670\obj\에서 확장자가 컴파일해서 생성된 파일로 TFT_LCD.hex 를 다운로드 한다. 카메라를 통해 TFTLCD에 이미지가 나오면 정상이다.

다음 아래는 OV7670를 2가지 수정하면 하드웨어 그대로 두고, 사용할 수 있다.

1) Ov7670.h 수정사항

```
#define CHANGE_REG_NUM 71
```

2) ov7670config.h 수정사항

```
const    char change_reg[CHANGE_REG_NUM][2]=
{

   {0x32,0x00},
   {0x2a,0x00},
   {0x11,0x83}, //83
   {0x12,0x46},//QVGA RGB565
   {0x12,0x46},

{0x42,0x7f},
   {0x4d,0x00},//0x09
   {0x63,0xf0},
   {0x64,0xff},
   {0x65,0x20},
   {0x66,0x00},
   {0x67,0x00},
   {0x69,0x5d},

   {0x13,0xff},
   {0x0d,0x41},//PLL
   {0x0f,0xc5},
   {0x14,0x11},
   {0x22,0xFF},//7f
   {0x23,0x01},
   {0x24,0x34},
   {0x25,0x3c},
   {0x26,0xa1},
   {0x2b,0x00},
   {0x6b,0xaa},
   {0x13,0xff},

   {0x90,0x0a},//
   {0x91,0x01},//
   {0x92,0x01},//
   {0x93,0x01},

   {0x94,0x5f},
   {0x95,0x53},
   {0x96,0x11},
```

```
    {0x97,0x1a},
    {0x98,0x3d},
    {0x99,0x5a},
    {0x9a,0x1e},

    {0x9b,0x00},//set luma
    {0x9c,0x25},//set contrast
    {0xa7,0x65},//set saturation
    {0xa8,0x65},//set saturation
    {0xa9,0x80},//set hue
    {0xaa,0x80},//set hue

    {0x9e,0x81},
    {0xa6,0x06},

    {0x7e,0x0c},
    {0x7f,0x16},
    {0x80,0x2a},
    {0x81,0x4e},
    {0x82,0x61},
    {0x83,0x6f},
    {0x84,0x7b},
    {0x85,0x86},
    {0x86,0x8e},
    {0x87,0x97},
    {0x88,0xa4},
    {0x89,0xaf},
    {0x8a,0xc5},
    {0x8b,0xd7},
    {0x8c,0xe8},
    {0x8d,0x20},

    {0x33,0x00},
    {0x22,0x99},
    {0x23,0x03},
    {0x4a,0x00},
    {0x49,0x13},
    {0x47,0x08},
    {0x4b,0x14},
    {0x4c,0x17},
    {0x46,0x05},
    {0x0e,0x75},        //    75 f5    night mode
    {0x0c,0x10},        //90    50    d0
};
```

### 3.9 JoyStick Mouse

프로그램은 구동은 9_JoyStickMouse\project내의 

파일인 JoyStickMouse.uvproj 더블클릭한다.
8_JoyStickMouse\project\Obj\test.hex 에서 확장자가 컴파일해서 생성된 파일로 test.hex 를 다운로드 한다. SW2 의 조이스틱을 상하좌우 움직이면 컴퓨터 화면의 마우스 커서가 조이스틱에 따라 이동하는 것을 알 수 있다.
확인방법은 장치관리자 USB 휴먼 인터페이스 장치가 [그림 3.8.1]처럼 하나 더 추가가 된다.

[그림 3.9.1] 장치관리자

[그림 3.9.2] 장치 등록 정보에 위치가 있는데 이곳을 수정 할 수도 있다.




[그림 3.9.2] 장치 등록 정보

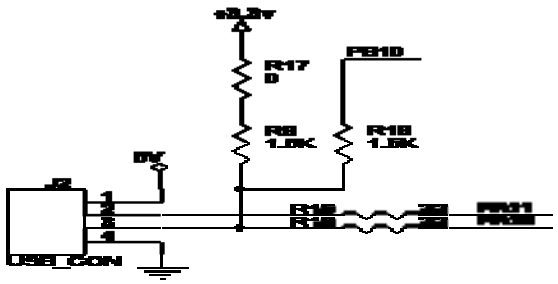
위치0( ) 내부를 수정 할려면 usb_desc.c 파일의 아래 함수부분에서 Signaltech-USB 수정하면 된다.

```
const u8 Joystick_StringProduct[JOYSTICK_SIZ_STRING_PRODUCT] =
  {
        JOYSTICK_SIZ_STRING_PRODUCT,          /* bLength */
        USB_STRING_DESCRIPTOR_TYPE,           /* bDescriptorType */

          'S', 0, 'i', 0, 'g', 0, 'n', 0, 'a', 0, 'l', 0, 'T', 0,
          'e', 0, 'c', 0, 'h', 0, '-', 0, 'U', 0, 'S', 0, 'B', 0
};
```

장치 인스턴트 id 관련 정보는 usb_desc.c 내 아래 함수부분을 보면 된다.

[그림 3.9.3] 장치 인스턴트 id

```
const u8 Joystick_DeviceDescriptor[JOYSTICK_SIZ_DEVICE_DESC] =
  {
   0x12,                          /*bLength */
  USB_DEVICE_DESCRIPTOR_TYPE, /*bDescriptorType*/
  0x00,                          /*bcdUSB */
  0x02,
  0x00,                         /*bDeviceClass*/
  0x00,                         /*bDeviceSubClass*/
  0x00,                         /*bDeviceProtocol*/
  0x40,                         /*bMaxPacketSize40*/
  0x83,                         /*idVendor (0x0483)*/
  0x04,
  0x10,                         /*idProduct = 0x5710*/
  0x57,
  0x00,                        /*bcdDevice rel. 2.00*/
  0x02,
   1,                  /*Index of string descriptor describing manufacturer */
   2,                  /*Index of string descriptor describing product       */
   3,                           /*Index of string descriptor describing the device serial number */    0x01
   /*bNumConfigurations*/
  }   ;                /* Joystick_DeviceDescriptor */
```




[그림 3.9.4] USB 관련 회로도

R8, R18 로 셋팅을 하는데, R8에 1.5K에, R18 저항 제거하면 프로그램없이 바로 사용하고, PB10을 다른 용도로 사용할 수 있다. R8에 저항을, R18에 1.5K 저항을 연결하면 프로그램으로 작성을 해야 만 한다. 이때 프로그램하지 않으면 인식이 절대 안 된다.
USB 를 사용할 때 D+를 Low 상태(USB 연결 안된 상태) =>초기화 상태 명령이 끝난 후 => D+ 풀 업이 되어야 인식이 된다. 이를 프로그램으로 확인은 hw_config.c 의 void USB_Cable_Config ( ) 에서 PB10을 H/L를 주면 되는데 아래를 보자.

```
void USB_Cable_Config (FunctionalState NewState)
{ if (NewState != DISABLE)
   {GPIO_SetBits(USB_DISCONNECT, USB_DISCONNECT_PIN); //PB 를 H
   }else
  {GPIO_ResetBits(USB_DISCONNECT, USB_DISCONNECT_PIN); } //PB 를 L
}
```

초기에 PB10 에 L 에서 PB10를 H 로 주면 D+ 풀 업이 되어 USB 가 인식이 된다.

## 3.10 uCOS

프로그램은 구동은 10_uCOS_DEMO\uCOSDemo.uvproj 더블클릭한다.
10_uCOS_DEMO\Obj\에서 확장자가 컴파일해서 생성된 파일로 uCOSDemo.hex를 다운로드 한다. TFTLCD 에 uCOS 데모 이미지가 나오면 정상이다.

### 3.10.1 uCOS (Real-Time)

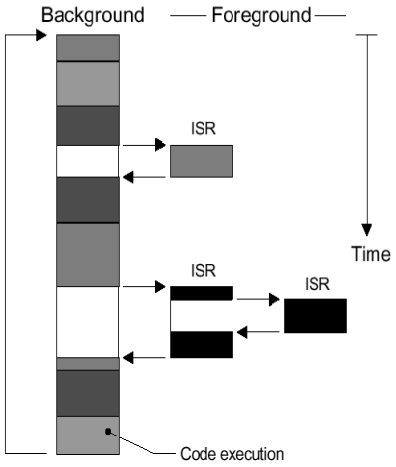
Real-Time이란 임의의 정보가 시스템에 입력 되었을 때 주어진 시간 안에 작업이 완료되어 결과가 주어져야 하는 것을 의미. 예측 가능하고 일정한 응답 시간을 요구하는 응용 프로그램의 지원을 위한 운영체제

- Hard Real-Time
  - Real-Time System 중에는 어떤 작업을 **일정 시간 안에 반드시 처리**해야 하며, 그 시간이 지난 후의 결과 값은 정확해도 소용이 없는 경우
    - Hard Real-Time System Ex) 군사장비, 비행기
- Soft Real-Time
  - 어떤 시간 안에 처리하면 좋지만 그렇지 못한 경우에 **그 시간에서 약간 경과한 후의 값도 인정**하는 경우
    - Soft Real-Time System Ex) cellular phone, router

Real-Time = Hard Real Time+ Soft Real Time

### 3.10.2 Background/ Foreground 시스템 (일반 마이크로 프로세서)

Background :Function을 이용하여 원하는 기능을 불러 프로그램을 무한 수행
Foreground : 인터럽트에 의해 ISR(Interrupt Service Routine)을 수행




### 3.10.3 크리티컬 섹션

**다른 TASK에 의해서 중단되어서는 안 되는 기능**을 말하며 공유자원의 임의 접근으로 충돌이 발생할 수 있기 때문에 주의를 해야 되고 Mutual exclusion이 보장되어야 한다. 이를 위해 인터럽트를 막거나 Semaphore를 사용하여 문제를 해결하고 있다.

### 3.10.4 자원과 공유자원

자원은 TASK가 사용하는 Entry(I/O, 변수, 메모리 등)를 뜻한다.
공유자원은 **한 개 이상의 TASK에 의해서 사용되는(공유되는) 자원을** 말하며 데이터의 손상을 방지하기 위해서는 공유 자원에 접근할 때 독점권을 얻어야 하는데 이를 상호배제(Mutual Exclusion)라고 한다.

### 3.10.5 멀티태스킹

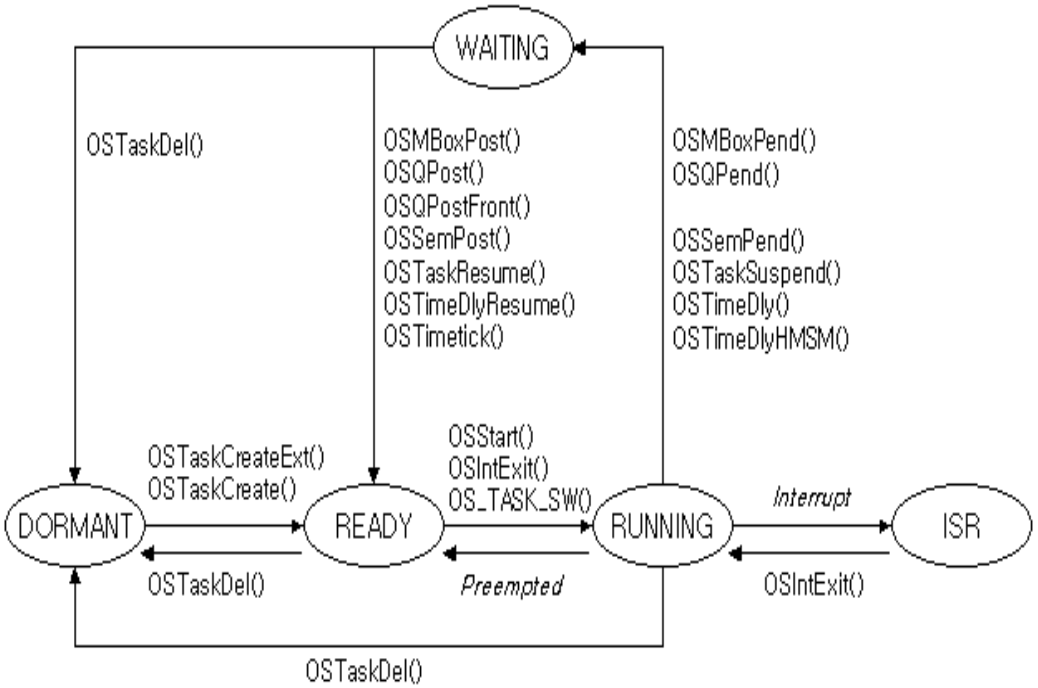
어떤 응용 프로그램이 수행되기 위해서는 여러 기능들이 동시에(물리적으로는 아님) 수행되어야 할 필요가 있고 이를 순차적으로 프로그램하기 어렵기 때문에 사용자가 프로그램을 쉽게 하기 위해서 도입한 개념이며 여러 가지의 작업이 동시에 이루어지는 것처럼 처리된다(실제로 1개의 CPU로는 한 순간에 하나의 일만 처리할 수 있다). CPU 사용률을 최대로 높이고 모듈 구조의 애플리케이션을 제공한다. 이를

이용하면 실시간 애플리케이션에서의 복잡성을 줄여주므로 디자인하기 편리하게 관리한다.

### 3.10.6 태스크

□1) Thread 2)하나의 간단한 프로그램으로 CPU를 사용하는 작업 3)각각의 우선순위, 스택 영역을 가짐

| 상 태 | 설 명 |
|---|---|
| DORMANT (수면) | 메모리에는 존재하지만, 수행될 수 있는 상태는 아님 |
| READY (준비) | 수행될 수는 있지만 현 태스크보다 우선순위가 낮을 때 |
| RUNNING (실행) | CPU를 실제로 사용하고 있을 때 |
| WAITING (대기) | Event의 발생을 기다릴 때 (예: I/O의 연산 종료, 공유 리소스가 사용 가능해 질 때, timing pulse 등) |
| ISR(인터럽트 서비스루틴) | 인터럽트가 일어나고 CPU가 인터럽트 서비스를 수행할 때 |



### 3.10.7 문맥전환 (Context Switching)

여러 TASK가 번갈아 가며 작업을 할 때 한 작업을 중단하고 다른 TASK를 수행하기 직전에 이전 TASK의 상태를 저장하고 새로운 TASK의 상태를 불러 들이는 교환 작업이 필요하다. 메모리 자원은 TASK마다 따로 할당하므로 문제가 없으나 CPU의 레지스터들은 공유 자원이므로 충돌을 방지하기 위해서는 현 상태 값을 저장해야 한다. Context Switching은 실제적인 프로세스 작업이 아니므로 이에 걸리는 시간은 Overhead이며 짧으면 짧을수록 좀 더 효율적인 운영체제가 된다.

### 3.10.8 커널

커널은 Multitasking 시스템의 한 부분으로, 태스크관리와 그 태스크 간의통신을 관리. 커널이 제공하는 가장 기본적인 서비스는 문맥전환(Context switch)이다. 보통 커널은 시스템에 약간의 부하(약 2～5%)를 주며 임베디드용의 경우 추가적으로 코드를 위한 ROM과  커널 데이터 구조체를 위한 RAM이 필요하다.
커널은 세마포어관리, 메일박스, 큐, 시간지연등으로 CPU를 더 편리하게 사용하도록 한다.

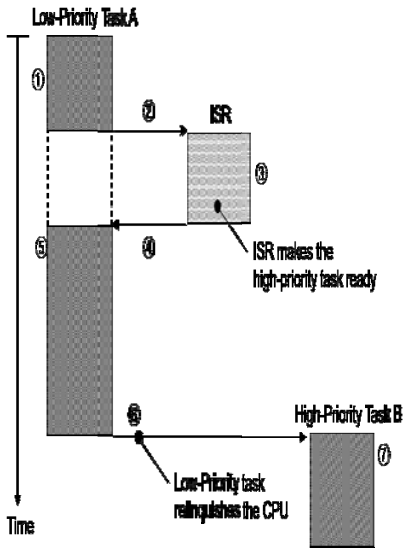
### 3.10.9 스케줄러

Multitasking환경에서 다음에 어떤 TASK를 수행할 것인지를 결정한다. 대부분 리얼타임은 우선순위를 기준. 우선순위는 응용프로그램에 따라 달라진다. 최상위 우선순위 태스크가 언제 CPU를 점유할것인가는 커널의 종류에 따라 다르다. 우선 순위 기반커널에는  비선점형과 선점형이 있다.

### 3.10.10 비선형 커널

어떤 TASK가 수행되고 있을 때 커널은 중간에서 그 TASK의 수행을 중지시키고 다른 TASK를 수행시킬 수 있는 능력이 없다.

처리순서
① Low priority taskA가 수행 중
② interrupt 발생
③ ISR에서 인터럽트 처를 하고 Scheduler를 호출하면 현 task보다 높은 priority를 가진 task를 READY 상태로 만듦
④ ISR의 수행이 종료되고 ISR 이전에 수행하던 Low priority taskA로 돌아감
⑤ 인터럽트 발생 직전의 코드부터 다시 수행
⑥ taskA가 system call을 호출하여 CPU 사용권 포기
⑦ kernel에 의해서 taskB가 수행

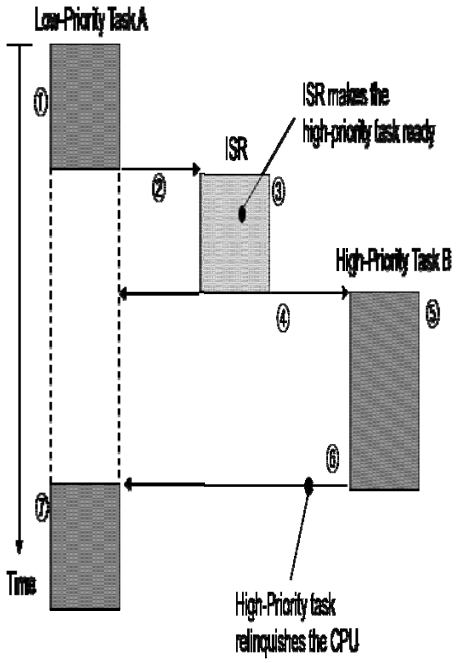


### 3.10.11 선점형 커널

어떤 TASK를 중단시키고 다른 TASK를 수행시킬 능력이 있으며 Interrupt latency가 길며, 구현이 어려운 대신 Priority를 반영할 수 있어 많이 사용된다.

처리순서

① Low priority taskA가 수행 중
② 인터럽트 발생
③ ISR에서 인터럽트 처리를 하고 Scheduler를 호출하여 현 task 보다 높은 priority를 가진 task를 READY 상태로 만듦
④ ISR 종료와 동시에 ISR 이전에 수행하던 taskA가 아닌 high priority taskB가 CPU 제어권을 가짐
⑤ taskB 수행
⑥ taskB가 kernel system call을 호출하여 CPU 제어권을 포기하고 kernel에 의해서 taskA가 선택됨
⑦ taskA의 인터럽트 발생 직전의 코드부터 다시 수행




### 3.10.12 재진입 가능 함수

코드가 재 진입할 수 있음을 나타내는 것으로 한 TASK에서 어떤 **함수를 수행하다가 도중에 다른 TASK로 바뀌어서도 이 함수를 호출했을 때 함수가 제대로 동작**할 수 있음을 나타낸다. 전역변수를 사용하지 않으면 이 문제를 쉽게 해결할 수 있으며, 공유 자원일 경우 Semaphore를 사용한다.

### 3.10.13 라운드 로빈 스케줄링 (타이밍 슬라이싱)

2개 이상의 태스크가 같은 우선 순위일 때, 커널은 퀀텀이라 불리는 미리 정해진 시간간격 동안 하나의 태스크를 실행한 뒤, 다른 한 태스크를 실행하는 과정을 말함

- 현재 태스크가 주어진 시간간격 동안 할 일이 아무것도 없을 때
- 현재 태스크가 주어진 시간간격 전에 끝났을 때
- 시간간격 전에 끝났을 때

### 3.10.14 태스크 우선 순위

각 태스크는 우선순위가 배정되어, **중요한 태스크일수록 더 높은 우선순위를 배정**한다. 커널을 사사용 할 때 프로그래머 스스로 적절한 우선순위를 지정한다.

### 3.10.15 정적 우선 순위 Static Priority

응용프로그램 실행 시, 태스크 **우선순위를 바꿀 수 없을 때** 태스크의 우선순위는 정적이다. 각 태스크는 컴파일 시 정적 우선순위가 배정된다. 우선순위가 정적인 시스템은 컴파일 시 모든 태스크의 시간 제약이 정해진다.

### 3.10.16 동적 우선 순위 Dynamic Priority

응용프로그램 실행 시, 태스크 **우선순위를 바꿀 수 있을 때** 태스크의 우선순위는

동적이다.
리얼타임 커널에서 우선순위 전도 문제를 해결. μC/OSⅡ는 이 기능을 지원.

**3.10.17 우선순위 전도 Priority Inversion**
Priority가 역전되는 현상으로 자원 할당을 위해 다음 TASK를 결정할 때 Priority에 따른 TASK 전환에 문제가 발생하게 된다. 이를 방지하기 위해 같은 Semaphore를 쓰는 TASK는 같은 Priority 를 주는 등의 방법을 사용한다.
**3.10.18 태스크 우선순위 배정**
실행빈도가 가장 높은 태스크에 최상위 우선순위를 배정 (RMS(Rate Monotonic Scheduling) 근거)
하드리얼타임 시간제약에 만족하려면 모든 태스크의 CPU 사용율이 70%이하가 돼야 한다.
RMS는 다음 몇가지를 가정
1. 모든 태스크는 주기적
2. 태스크는 서로 동기되지 않고, 자원을 공유하지 않고, 데이터를 주고 받지 않는다.
3. CPU는 항상 실행 준비상태가 된 최상위 우선순위의 태스크를 실행

**3.10.19 Mutual Exclusion**
**하나의 task가 공유자원을 사용하고 있는 동안 다른 task가 이 자원을 사용하지 못하도록 보장**
태스크가 가장 쉽게 서로 정보 교환할수 있는 방법은 공유 데이터를 통해서 이다.
공유자원의 독점적인 액세스를 얻는 가장 일반적인 방법
1. 인터럽트 비활성화
2. Test-And-Set(TAS) 수행
3. 스케줄링 비활성화
4. 세마포어 활성화

뮤텍스 같은 동기화 오브젝트의 경우 한 번에 사용중인 모든 스레드를 수행하거나 아니면 한 번에 하나의 스레드를 수행한다. 각 태스크는 테이터 손실을 방지하기 위해 독점적으로 고유 자원을 액세스 해야 하는 데 이것을 뮤텍스 라 한다.
뮤텍스는 여러 면에서 크리티컬 섹션과 비슷하고, 대신 사용할 수도 있지만 이름을 가질 수 있다는 점에서 크리티컬 섹션보다 우월하다.

**3.10.20 인터럽트 비활성화**
공유자원의 독점적인 액세스를 얻는 가장 빠른 방법

```
OS_ENTER_CRITICAL();                    //인터럽트 비활성화
. . . .
OS_EXIT_CRITICAL();                     //인터럽트 활성화( 앞에
OS_ENTER_CRITICAL(); 있어야 한다)
```

μC/OSⅡ는 내부변수와 데이터 구조체를 액세스 하기 위해 서 이 방법을 사용.
**3.10.21 Test-And-Set(TAS) 수행**
커널을 사용하지 않고 자원을 액세스 하려면, 함수가 자원을 액세스하기 전에 전역변수를 점검하고 그 변수가 0일 때만 그 자원에 액세스한다는 법칙을 따라야 한다. **다른 함수가 그 자원을 액세스하는 것을 막기 위해 자원을 액세스하는 함수가 그 변수의 값을 1로 세트해 주면 된다.** TAS 오퍼레이션은 마이크로프로세서가 인터럽

트에 의해 선점될 수 없게 실행돼야 한다.

인터럽트 비활성화;
IF('액세스 변수'가 0인가)
**액세스 변수의 값을 1로 설정**
인터럽트를 활성화한다;
자원을 액세스해서 사용한다;
인터럽트를 비활성화한다;
액세스 변수를 0으로 세팅한다;
인터럽트 활성화;
} else {
인터럽트 활성화;
(자원을 공유할수 없다)
}

### 3.10.22 스케줄링 비활성화와 활성화

스케줄링을 비활성화하고 활성화할 수 있다. 이 경우 둘 이상의 태스크가 경쟁 없이 데이터를 공유할 수 있다. 스케줄러가 잠겨있는 동안에도 인터럽트는 활성화돼 있으므로 크리티컬 코드를 실행하는 도중 인터럽트가 발생하면 ISR이 즉시 실행된다는 것

### 3.10.23 세마포어

세마포어는 1960년대 중반에 에드거 다익스트라(Edsger Dijkstra)에 의해 발명되어졌다. 신호등의 역할을 한다. 현재 실행중인 Thread가있으면 빨간 불을 켜서 다른 Thread가 들어오지 못하도록 하는 것이다. 세마포어의 경우 최대 개수를 지정하여 동시에 몇 개의 스레드가 특정 작업을 수행할 수 있는지를 지정해서 사용하는 방식. 세마포어란 역시 데드락을 피하기 위한 기술. 멀티태스킹 시스템에서는 없어서는 안될 중요한 자료구조 인데, 두 가지로 구분한다. 하나는 바이너리(0,1) 세마포어(Binary Semaphore), 그리고, 다른 하나는 카운팅(8,16,32비트 사용) 세마포어(Counting Semaphore)이다.

카운팅 세마포어 : 카운팅 세마포어는 내부 카운트를 사용하기 때문에 여러 차례 획득하고 반할 수 있다. 카운팅 세마포어를 생성할 때 세마포어의 초기 토큰 개수를 지정할 수 있다. 초기 카운트 값을 0 으로 지정하면 세마포어는 사용 불가능 상태로 생성되며, 카운트 값을 1 이상으로 지정할 경우 세마포어는 사용가능 상태로 생성된다.

바이너리 세마포어 : 바이너리 세마포어는 0 이나 1 의 값을 갖는다. 바이너리 세마포어의 값이 0 일 때 세마포어는 사용 불가능하며, 반대로 값이 1 때는 세마포어가 사용 가능하다.

세마포어들은 일반적으로 메모리 공간을 공유하거나, 또는 파일들을 공유 액세스하기 위한, 두 가지 정도의 목적을 위해 사용된다.
Semaphore 용도
• 공유 자원간의 액세스 제어
• 이벤트 발생을 알려줌
• 두 TASK사이에 동기화 를 맞추는 데도 사용
INITIALIZE, WAIT, SIGNAL이라는 세 가지의 동작을 수행하게 되는데, INITIALIZE의

경우는 처음 세마포어의 값을 초기화시켜 주는 함수이다. WAIT는 세마포어 획득하고자 하는 태스크이고, SIGNA는 세마포어를 양도한다.
공유자원 액세스 제어용으로 사용할 경우 열쇠기호, 이벤트 발생 신호로 사용할 경우 깃발로 표현한다.

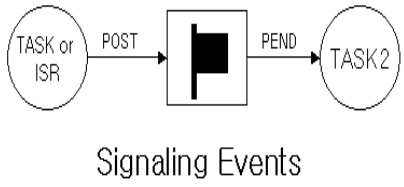
### 3.10.24 교착상태 Deadlock

두 태스트가 각자 다른 태스크에서 쓰고 있는 자원을 무한정 기다리는 상태.
시스템이 더 이상 작업을 처리할 수 없는 상태에 도달하는 것으로 서로가 상대의 자원 해제를 기다리는 등의 악순환으로 대기 상태가 지속된다. 이도 역시 Semaphore를 이용하면 풀 수 있다.
1) 프로세스를 진행하기전 필요한 모든 자원 획득
2) 순서대로 자원 획득
3) 역순으로 자원 양도

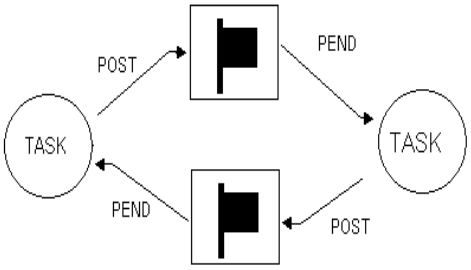
### 3.10.25 동기화

인터럽트 서비스 루틴이나 태스크에서 이벤트를 발생하게 되면, 그 이벤트를 기다리고 있던 태스크의 수행을 계속시키는 동기화 방법이다.

- 이벤트를 기다리는 최상위 우선순위의 태스크 신호를 보낸다.
- 이벤트를 가정 먼저 기다리고 있는 태스크에게 신호를 보낸다.



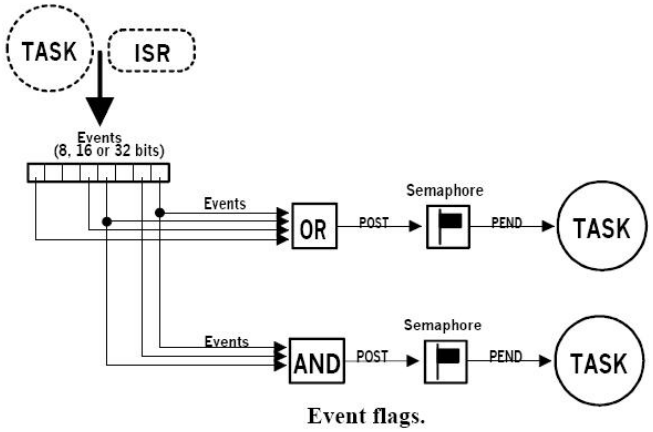
Signaling Events




태스크가 서로서로 동기화 된다는 것에 차이가 있다. 예로, 첫 번째 태스크가 특정 작업을 수행하고, SIGNAL을 두 번째 태스크에게 전달한다. 그리고, WAIT을 수행해 두 번째 태스크로부터 SIGNAL을 기다린다. 두 번째 태스크 역시 특정 작업을 수행한 후, 첫 번째 태스크로 SIGNAL을 전달하고, WAIT을 통해 첫 번째 태스크로부터 SIGNAL을 기다리는 동기화 방법이다. 한 가지 주의할 점은 태스크간의 동기화에만 사용될 수 있다는 것이다.

### 3.10.26 이벤트 플래그

이벤트 플래그는 어떤 태스크가 다수 개의 이벤트 발생에 대해 동기화가 필요할 때 사용한다. 여기도 두 가지 방법이 있을 수 있는데, 하나는 분리 동기화(Disjunctive Synchronization)이고, 다른 하나는 결합 동기화(Junctive Synchronization)이다. 분리 동기화는 다수 개의 이벤트 중에서 만족하는 하나의 이벤트라도 발생할 경우에는 동기화가 성립된다. 논리회로에서 OR연산과 유사하다. 결합 동기화는 필요한 이벤트 모두가 발생했을 경우에만 바라는 동기화가 성립하는 경우이다. 논리회로에서의 AND연산과 유사한 개념이다.
플래그 설정시 부수적 조건으로 AND와 OR 두가지 경우가 정의되어 있다. AND인 경우, 모든 비트값이 정확히 일치해야 하고 OR인 경우에는 한 비트만 일치하더라도 대기 상태가 해제되게 된다.



Event flags.

이벤트 플래그를 지원하는 커널은 이벤트 플래그를 set, Clear, Wait 서비스를 제공한다. 이벤트 플래그는 플래그에 특정한 값이 설정될 때 까지 태스크들을 대기시키는 용도로 사용되는데, 설정되는 값은 비트 필드(bit field)로 이루어 진다.

세마포어가 주로 상호배제에 촛점을 맞추어 만들어진 것이라면 이벤트 플래그는 태스크간 동기화에 중점을 둔 것이라 할 수 있다.
세마포어로 태스크 동기화 처리가 불가능한 것이 아니며, 마찬가지로 이벤트 플래그로 상호배제를 구현할 수 도 있다. 다만 어디에 촛점을 두었는가에 따라 적용의 용이함이나 편의성에서 차이가 난다.
동기화에 따른 차이점으로 세마포어는 간단한 경우, 정보를 1개만 전달, Event Flags는 복잡한 경우, 2개 이상의 정보를 전달할 때 사용

**3.10.27 TASK Communication**
TASK간의 통신을 위한 방법으로는 Global variable(전역 변수)를 쓰는 것과 Message passing의 2가지 방법이 있다. Global variable을 쓰는 경우에 Exclusive access를 해야 하는데
ISR(Interrupt Service Routine)이 포함된 Interrupt를 Disable해야 하며 TASK 사이에서는 Interrupt 를 Disable하는 방법 외에 Semaphore를 쓸 수도 있다.
태스크가 변수의 값이 변경 된 것을 알 수 있는 것은 변수를 모니터링하거나, ISR이 세마포어를 이용해서 태스크로 신호를 보내는 것이다. 이러한것을 해결하기 위해 Mailbox, Queue사용
Message passing의 방법으로는 Mailbox, Queue,Pipe등이 있다.

**2.23 Message Mailboxes**
Message는 커널이 제공하는 서비스에 의해 TASK로 전달된다. Message Exchange라고도
불리며, TASK는 전달받은 Message나 Message 포인터 값을 인식할 수 있다.
커널은 다음과 같은 메일박스 서비스를 제공

- 메일박스의 내용을 **초기화**. 초기에 메시지를 포함 또는 안 할 수도 있다.
- 메일박스에 메시지를 **송부**한다.(POST)
- 메일박스에 메시지가 **도착**하기를 기다린다.(PEND)
- 메일박스에 메시지가 있으면 **가져온다**. (ACCEPT)

**3.10.28 Message Queues**

TASK에 1개 **이상**의 메시지를 전달할 때 사용하며 메일박스가 **여러 개** 있는 것으로 볼 수 있다. 여러 TASK의 메시지가 **FIFO**(Queue)에 넣고 빼다.
커널은 다음과 같은 큐 메시지 큐 서비스를 제공

- 큐 를 **초기화**. 큐는 항상 초기화 후 비어있다고 가정.
- 큐에 메시지를 **송부**한다.(POST)
- 큐 메시지가 **도착**하기를 기다린다.(PEND)
- 큐 메시지가 있으면 **가져온다**. (ACCEPT)

OSQPost()=>큐에 메시지 보내기(FIFO)
OSQPostFront()=>큐에 메시지 보내기(LIFO)

**3.10.29 Interrupt**

Asynchronous event를 CPU에 알리는 방법으로써 Interrupt latency(지연시간)가 중요하며 ISR(Interrupt Service Routine)에서 처리한다. 인터럽트는 가능한 짧게 비활성화 돼야 한다. 프로세서가 인터럽트를 수행하는 동안 다른 인터럽트를 인식해서 수행.

**3.10.30 Interrupt latency(지연시간)**

인터럽트에 반응해 서비스 루틴을 수행하기까지의 시간

```
void   TaskStart (void *pdata)
{
TaskStartDispInit();
OS_ENTER_CRITICAL();              //인터럽트 비활성화
……..……..
  OS_EXIT_CRITICAL();             //인터럽트 활성화( 앞에 OS_ENTER_CRITICAL(); 있어야 한다)
}}
```

Interrupt 비활성화가 길어질수록 더 길어진다.
Interrupt latency = Interrupt 가 비활성화된 최대시간 + ISR 에서 최초 명령을 시작하려는 시간

**3.10.31 Interrupt Response(응답시간)**

인터럽트 서비스 루틴으로 분기하기 전에, 현재 태스크의 내용들을 저장하는 작업이 필요해 진다. 선점형 커널인 경우는 인터럽트 내포 처리를 위해 여분의 작업 시간이 더 필요해 진다. 응답시간(Response Time)을 계산하는 식은 아래와 같다.

**비선점형 커널인 경우,**
**응답시간 = 인터럽트 지연 + 현재 태스크의 내용을 저장하는 시간**

**선점형 커널인 경우,**
**응답시간 = 인터럽트 지연 + 현재 태스크의 내용을 저장하는 시간**
**+ 인터럽트 서비스 루틴 진입 함수를 수행하는 시간**

**3.10.32 인터럽트 복귀시간**

인터럽트 복귀시간은 **인터럽트된 코드로 다시 복귀하는데 걸리는 시간**을 말한다. 선점형 커널의 경우 인터럽트 내포을 고려해야 하므로, 가장 높은 우선 순위를

선택하는 시간이 추가적으로 포함된다.

비선점형 커널인 경우,
인터럽트 복귀시간 = CPU 내용을 되돌리는 시간
+ 인터럽트된 태스크로 돌아가는 시간
선점형 커널인 경우,
인터럽트 복귀시간 = CPU 내용을 되돌리는 시간
+ **가장 높은 우선 순위의 태스크를 선택하는 시간**
+ 인터럽트된 태스크로 돌아가는 시간

### 3.10.33 ISR 수행시간

인터럽트 서비스 루틴의 처리 시간 역시 고려하는 것이 필요한데, 가능한 짧게 구성하는 것이 좋지만, 절대적인 기준은 없다. 하지만, 대부분의 경우 인터럽트 서비스 시, 실제 작업은 태스크에서 수행하게 한다. 이를 위해, 앞에서 언급한, 태스크 간의 통신 방법들, 즉, 세마포어, 메시지 박스, 메시지 큐를 사용하게 되는데, 이들이 수행되는 시간 역시 만만치가 않다. 그래서, 간단한 경우, 인터럽트 서비스 루틴 내부에서 직접 수행을 시키는 것이 바람직하다.

### 3.10.34 Non-maskable Interrupt

NMI(Non-maskable Interrupt) 역시 알아둘 필요가 있는데, 인터럽트가 가능한 빨리 처리되어야 할 경우 유용한 방법이다. NMI는 막을 방법이 없기 때문에, 인터럽트 지연, 응답 시간, 복귀시간이 최소이다. 원래는 파워가 갑자기 끊길 경우를 위해 마련된 인터럽트지만, 일반적인 상황에서도 사용할 수 있다.

NMI의 경우,
인터럽트 지연시간 = 가장 긴 명령어 실행 시간
+ NMI ISR 실행을 시작하는데 걸린 시간
인터럽트 응답시간 = 인터럽트 지연 + CPU 내용을 저장하는데 걸린 시간

인터럽트 복귀시간 = CPU 내용을 복귀시키는데 걸린 시간
+ 인터럽트된 태스크로 돌아가는데 걸린 시간

NMI는 무시될 수 없는 인터럽트기 때문에, 임계 영역의 진입을 막을 수 있는 방법이 없다. 그러므로, 태스크를 깨우는데 필요한 태스크 통신 서비스들은 사용할 수가 없게 된다. 하지만, 전역 변수를 사용한 방법은 유효하기 때문에, 이를 사용할 경우 태스크와 ISR 간의 정보 교환이 가능해 진다.

**3.10.35 Clock Tick**

주기적으로 발생하는 특별한 인터럽트로 시스템의 심장 역할을 하며 시스템의 기준 시간이다.

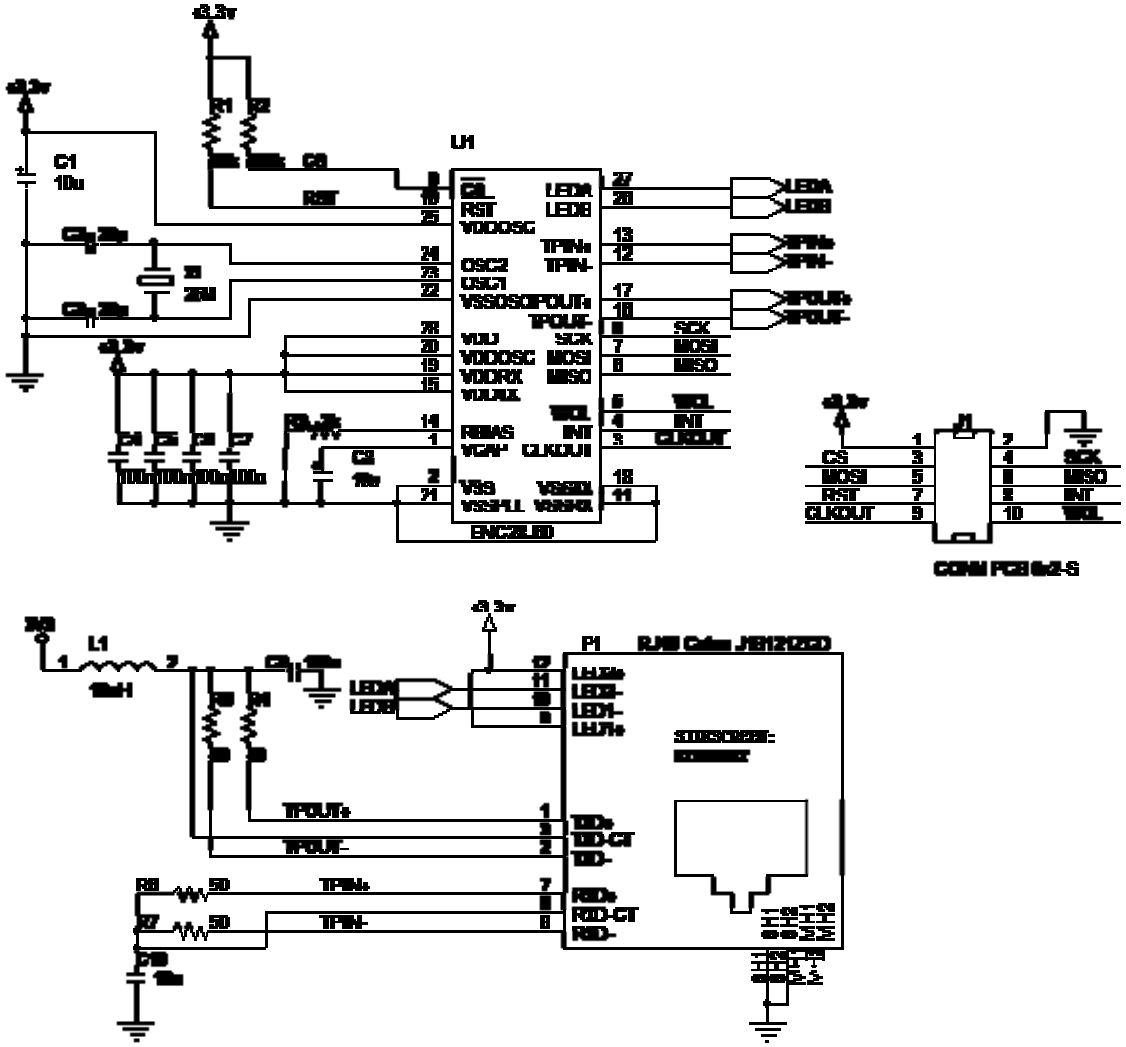
uC/OS-II는 실시간 커널이다. 실시간 시스템은 각 태스크의 데드라인을 중요한 요소로 하며 실시간 OS는 실시간 시스템을 관리하는 것이 목적이므로, 당연히 실시간 OS도 데드라인을 중요한 변수로 취급한다. 그리고 실시간 OS는 태스크들을 시간을 변수로 관리하기 위한 몇 가지 함수를 갖추고 있으며, 실시간 커널인 uC/OS-II 역시 같은 기능을 제공하는 함수를 갖추고 있다.

uC/OS-II는 클락을 이용하여 시간을 관리하며, OSTimeDly(), OSTimeDlyHMSM(), OSTimeDlyResume(), OSTimeGet(), OSTimeSet() 함수를 제공한다. 이 함수들은 OS_TIME.C에 선언되어 있다.

## 3.11 ENC28J60

1) 하드웨어 구성

STM32 보드와 ENC28J60보드 연결
PA4 =>NSS, PA5=> SCK, PA6=>MISO, PA7=> MOSI
STM32 보드의 PA0, PA1에 LED+ 연결하고 LED- 는 GND 연결한다.
PC 의 랜선을 ENC28J60보드에 연결한다.




ENC28J60보드 회로도

2) PC 설정

시작 명령 창에서 cmd 쓰면 도스창이 생성되고, 도스 창에서 띄워 연결 상태를 알기 위해 ping 192.168.0.3 –t 한다.
아래와 같은 형태로 응답을 하면 연결이 된 상태이다.

제어판 => 네트워크 상태 및 작업 보기

로컬 영역 연결

속성 선택

Internet Protocol Version 4 (TCP/IPv4) 더블 클릭한다.

아래와 같이 설정한다. 그리고 확인을 클릭한다.

3) 웹 실행

Internet Explorer를 실행한다.
주소창에 아래와 같이 기입한다.
http://192.168.0.2/123456

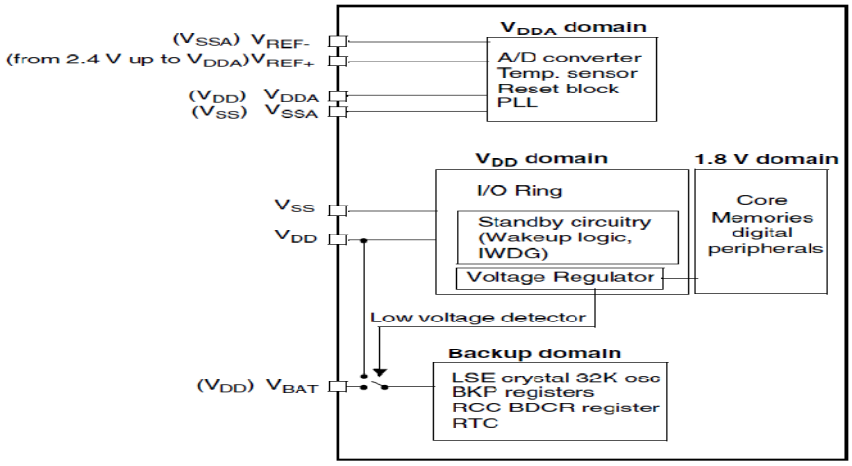
아래 화면처럼 나타난다. LED ON 을 클릭하면 보드의 LED에 LED 가 ON 되고 다시 한번 클릭하면 OFF가 된다.

# 제 4 장 STM32 내부기능

STM32 내부 구조에 대해서 설명한다.

## 4.1 PWR(Power control)

디바이스가 요구하는 operating voltage supply (VDD)는 2.0 ~ 3.3V 까지 사용한다. 자체 레귤레이터가 내부 1.8V 디지털 전원을 공급해 사용한다. 일반 I/O 포트 2V~5.5V 사용 가능하다.
메인 VDD 전원이 차단되어도 RTC (real-time clock) 와 backup registers 는 VBAT voltage(1.8~3.6V) 에 의해 전원이 공급된다.




위 그림에서 64핀 경우이며 VREF+ 과 VREF-핀은 사용하지 않고, ADC 전압은 + (VDDA) 과 GND(VSSA) 내부적으로 연결되어 있다. 100핀 경우 VREF+과 VREF-핀은 ADC 입력전압 (2.4 V ~ VDDA)으로 사용한다.

1) Low-power modes

| Mode name | Entry | wakeup | Effect on 1.8V domain clocks | Effect on $V_{DD}$ domain clocks | Voltage regulator |
|---|---|---|---|---|---|
| Sleep (Sleep now or Sleep-on - exit) | WFI | Any interrupt | CPU CLK OFF no effect on other clocks or analog clock sources | None | ON |
| | WFE | Wakeup event | | | |
| Stop | PDDS and LPDS bits + SLEEPDEEP bit + WFI or WFE | Any EXTI line (configured in the EXTI registers) | All 1.8V domain clocks OFF | HSI and HSE oscillators OFF | ON or in low-power mode (depends on *Power control register (PWR_CR)*) |
| Standby | PDDS bit + SLEEPDEEP bit + WFI or WFE | WKUP pin rising edge, RTC alarm, external reset in NRST pin, IWDG reset | | | OFF |

STM32F103 디바이스 특성 (3 가지 low-power modes)
- Sleep mode (Cortex-M3 core stopped, peripherals kept running)
- Stop mode (all clocks are stopped)
- Standby mode (1.8V domain powered-off)

- Run Mode 는 1.8V 를 Core, Memory, digital peripherals 에, 최대전력공급
- Stop Mode 는 1.8V 를 레지스터, SRAM 에 적은 전력공급
- Standby Mode 가장 적은 전력소모

2) Sleep Mode

CPU 코아만 정지하다가 WFI (Wait For Interrupt), WFE (Wait for Event)가 되면 깨어나 동작하는 상태. 8MHZ 일 때 7.5mA 에서 5.5mA 줄어든다.

3) Stop Mode

| Stop mode | Description |
|---|---|
| **Mode entry** | WFI (Wait for Interrupt) or WFE (Wait for Event) while:<br>– Set SLEEPDEEP bit in Cortex™-M3 System Control register<br>– Clear PDDS bit in Power Control register (PWR_CR)<br>– Select the voltage regulator mode by configuring LPDS bit in PWR_CR<br><br>**Note:** To enter Stop mode, all EXTI Line pending bits (in *Pending register (EXTI_PR)*) and RTC Alarm flag must be reset. Otherwise, the Stop mode entry procedure is ignored and program execution continues. |
| **Mode exit** | If WFI was used for entry:<br>Any EXTI Line configured in Interrupt mode (the corresponding EXTI Interrupt vector must be enabled in the NVIC). Refer to *Table 27: Vector table on page 97*<br>If WFE was used for entry:<br>Any EXTI Line configured in event mode. Refer to *Section 6.2.3: Wakeup event management on page 101* |
| **Wakeup latency** | HSI RC wakeup time + regulator wakeup time from Low-power mode |

4) Standby Mode

| Standby mode | Description |
|---|---|
| **Mode entry** | WFI (Wait for Interrupt) or WFE (Wait for Event) while:<br>– Set SLEEPDEEP in Cortex™-M3 System Control register<br>– Set PDDS bit in Power Control register (PWR_CR)<br>– Clear WUF bit in Power Control/Status register (PWR_CSR) |
| **Mode exit** | WKUP pin rising edge, RTC alarm, external Reset in NRST pin, IWDG Reset. |
| **Wakeup latency** | Regulator start up. Reset phase |

Standby Mode는 낮은 전력 소비를 위해서 이다.
PLL, HSI와 HSE RC oscillators 는 switched off 이다.

5) Power control registers

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000h | PWR_CR | Reserved | | | | | | | | | | | | | | | | | | | | | | | DBP | PLS[2:0] | | | PVDE | CSBF | CWUF | PDDS | LPDS |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 004h | PWR_CSR | Reserved | | | | | | | | | | | | | | | | | | | | | | | EWUP | Reserved | | | | | PVDO | SBF | WUF |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | 0 | | | | | | 0 | 0 | 0 |

PWR register map

- Power control register (PWR_CR)

**Bits 31:9** Reserved, always read as 0.

**Bit 8** DBP: *Disable Backup Domain write protection*.
In reset state, the RTC and backup registers are protected against parasitic write access. This bit must be set to enable write access to these registers.
0: Access to RTC and Backup registers disabled
1: Access to RTC and Backup registers enabled

**Bits 7:5** PLS[2:0]: *PVD Level Selection*.
These bits are written by software to select the voltage threshold detected by the Power Voltage Detector
000: 2.2V 001: 2.3V 010: 2.4V 011: 2.5V
100: 2.6V 101: 2.7V 110: 2.8V 111: 2.9V
Note: Refer to the electrical characteristics of the *datasheet* for more details.

**Bit 4** PVDE: *Power Voltage Detector Enable*.
This bit is set and cleared by software.
0: PVD disabled 1: PVD enabled

**Bit 3** CSBF: *Clear Standby Flag*.
This bit is always read as 0.
0: No effect 1: Clear the SBF Standby Flag (write).

**Bit 2** CWUF: *Clear Wakeup Flag*.
This bit is always read as 0.
0: No effect 1: Clear the WUF Wakeup Flag after 2 System clock cycles. (write)

**Bit 1** PDDS: *Power Down Deepsleep*.
This bit is set and cleared by software. It works together with the LPDS bit.
0: Enter Stop mode when the CPU enters Deepsleep. The regulator status depends on the LPDS bit.
1: Enter Standby mode when the CPU enters Deepsleep.

**Bit 0** LPDS**:** Low-power deepsleep.
This bit is set and cleared by software. It works together with the PDDS bit.
0: Voltage regulator on during Stop mode
1: Voltage regulator in low-power mode during Stop mode

- Power control/status register (PWR_CSR)

**Bits 31:9** Reserved, always read as 0.

**Bit 8** EWUP: *Enable WKUP pin*
This bit is set and cleared by software.
0: WKUP pin is used for general purpose I/O. An event on the WKUP pin does not wakeup the device from Standby mode.
1: WKUP pin is used for wakeup from Standby mode and forced in input pull down configuration
(rising edge on WKUP pin wakes-up the system from Standby mode).
Note: This bit is reset by a system Reset.

**Bits 7:3** Reserved, always read as 0.

**Bit 2** PVDO: *PVD Output*
This bit is set and cleared by hardware. It is valid only if PVD is enabled by the PVDE bit.
0: VDD is higher than the PVD threshold selected with the PLS[2:0] bits.
1: VDD is lower than the PVD threshold selected with the PLS[2:0] bits.
Note: The PVD is stopped by Standby mode. For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.

**Bit 1** SBF: *Standby Flag*
This bit is set by hardware and cleared only by a POR/PDR (Power On Reset/Power Down Reset) or by setting the CSBF bit in the *Power control register (PWR_CR)*
0: Device has not been in Standby mode
1: Device has been in Standby mode

**Bit 0** WUF: *Wakeup Flag*
This bit is set by hardware and cleared only by a POR/PDR (Power On Reset/Power Down Reset) or by setting the CWUF bit in the *Power control register (PWR_CR)*
0: No wakeup event occurred
1: A wakeup event was received from the WKUP pin or from the RTC alarm
Note: An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.

## 4.2 RCC

RCC(Reset Clock Controller)는 STM32 의 Reset 과 내부 Clock 을 컨트롤 한다.
STM32 의 Clock 은 System Clock 과 Secondary Clock 으로 구분한다.

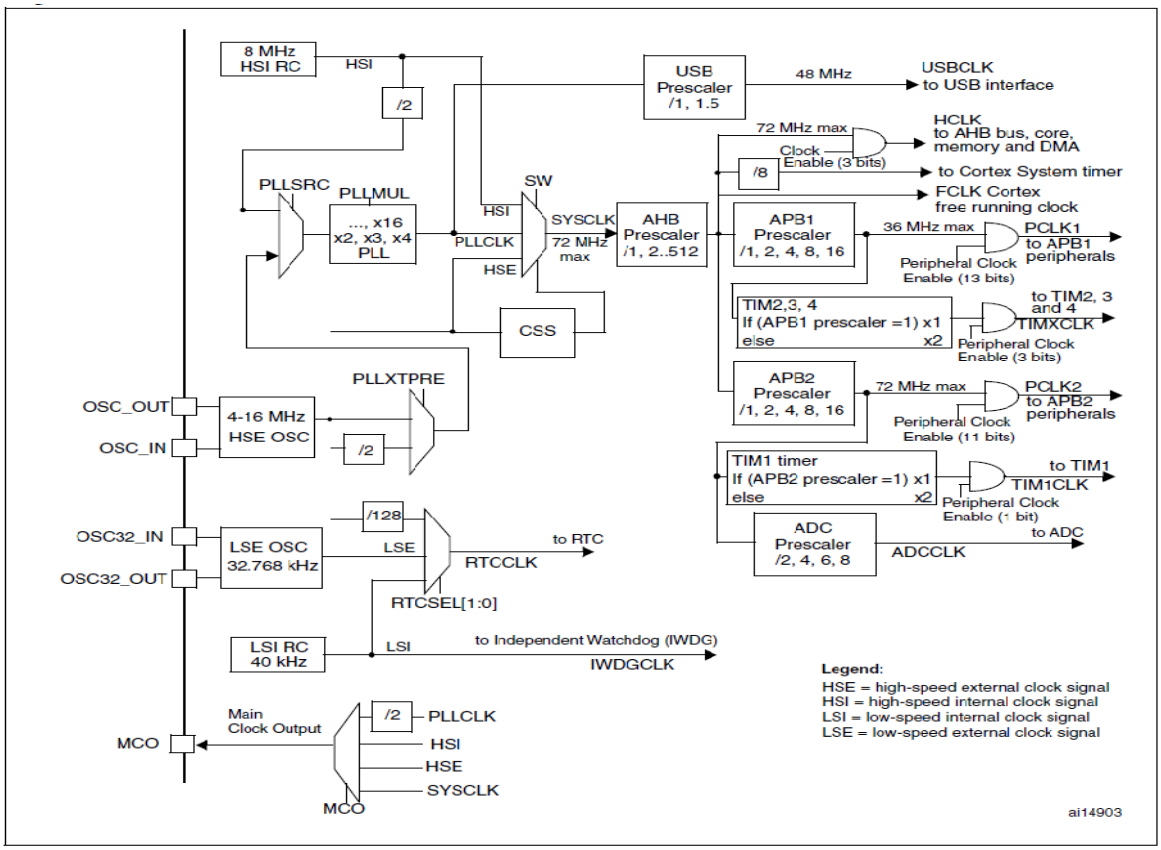
### 4.2.1 System Clock(SYSCLK)

SYSCLK 는 AHB, APB 에 연결된 각 Controller 에 공급되는 Main Clock 이다.
SYSCLK 에서 사용되는 Clock Source 는 다음 3 가지이다.
1) HSI (8MHz)내부 8MHz RC 클럭
2) HSE(4~16Mhz)외부 Oscillator Clock
3) PLLCLK(~72Mhz)PLLSRC 에서 선택된 HSI 또는 HSE 를 PLLMUL 에 의해서 분주한 Clock

### 4.2.2 Secondary Clock

RTCCLK 나 Watchdog 에서 사용되는 Clock 이며, Clock 소스는 2 가지이다.
1)LSI RC(40kHz) Low power Clock 소스
2)LSE OSC(32.768kHz) RTCCLK 소스로 사용.

### 4.2.3 System Clock Source 의 흐름을 아래와 같다.




STM32 라이브러리의 Default 설정은 HSE 8MHz 로 72MHz 의 SYSCLK 를 사용한다.
HSE Clock 이 Line 을 따라서 PLLMUL 에서 9 배로 되어 72MHz 가 된다.

1) PLLXTPRE 에서 HSE 를 Bypass 하거나 1/2 분주 할 수 있다. 72MHz 클럭을 만들기 위해서 Bypass 를 선택한다.

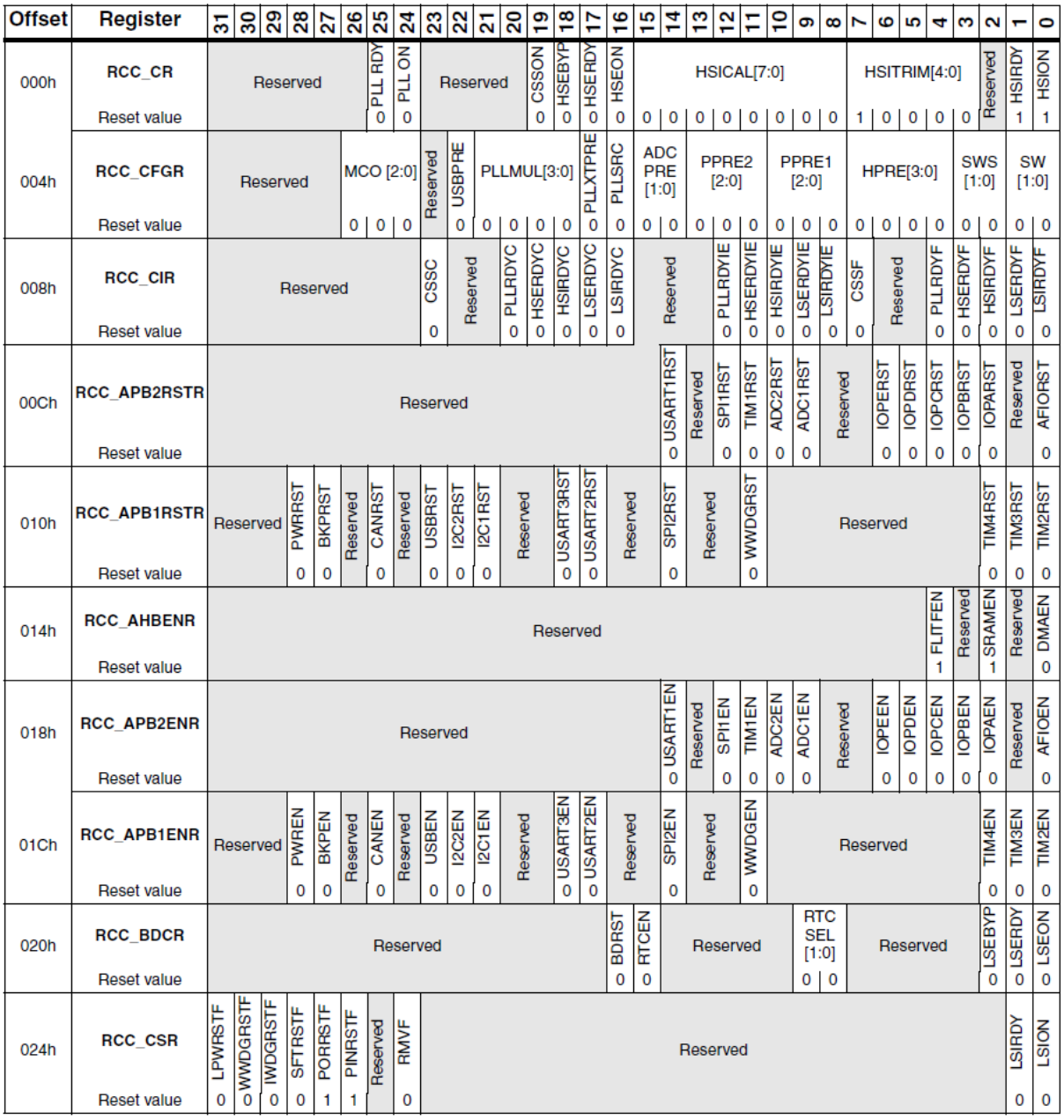
2) PLLSRC 에서 HSI 와 HSE 클럭 소스를 선택해서 PLLMUL 로 연결 시켜준다. PLLMUL 에서는 입력된 클럭 소스를 배수로 증가시키는 동작을 한다. PLLMUL 에서는 x4 ~ x9 배까지 클럭을 증가시킬 수 있다. 이때 PLL 출력 클럭이 72MHz 이내이다. 8MHz 를 72MHz 로 만들기 위해서 x9 를 선택한다.

3) SW(System Clock Switch)에서 HSI, HSE, PLLCLK 중에서 SYSCLK 로 사용할 클럭을 설정한다. 72MHz 의 PLLCLK 를 선택한다.

SYSCLK 는 AHB Prescaler 에 의해서 AHB 버스의 HCLK 로 사용되며, HCLK 는 각 내부 시스템의 FCLK, PCLK1, PLCK2, TIMxCLK, ADCCLK 등의 Clock Source 로 사용이다.
내부 시스템은 모두 자신의 Prescaler 를 사용해서 Clock 설정이 가능하며, 그 중 타이머에 사용되는 TIMxCLK 는 PCLK 의 Prescaler 값이 1 일 경우에는 bypass 로 연결되고, 그 외의 값일 경우에는 PCLK 의 x2 배의 clock 이 TIMxCLK 로 연결이다.

| Clock | 속도 | 비고 |
|---|---|---|
| SYSCLK | ~ 72MHz | SW 에 의해서 선택된 Source |
| HCLK | ~ 72Mhz | SYSCLK / AHBPrescaler |
| PCLK1 | ~ 36Mhz | HCLK / APB1Prescaler |
| PCLK2 | ~ 72Mhz | HCLK / APB2Prescaler |
| TIMxCLK | ~ 72Mhz | If(APBxPrescaler == 1) PCLKx*1<br>else PCLKx*2 |
| ADCCLK | ~ 72Mhz | PCLK2 / ADCPrescaler |

### 4.2.4 RCC Register Map



| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000h | RCC_CR | Reserved | | | | | | PLL RDY | PLL ON | Reserved | | | | CSSON | HSEBYP | HSERDY | HSEON | HSICAL[7:0] | | | | | | | | HSITRIM[4:0] | | | | | Reserved | HSIRDY | HSION |
| | Reset value | | | | | | | 0 | 0 | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | | 1 | 1 |
| 004h | RCC_CFGR | Reserved | | | | | MCO [2:0] | | | Reserved | USBPRE | PLLMUL[3:0] | | | | PLLXTPRE | PLLSRC | ADC PRE [1:0] | | PPRE2 [2:0] | | | PPRE1 [2:0] | | | HPRE[3:0] | | | | SWS [1:0] | | SW [1:0] | |
| | Reset value | | | | | | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 008h | RCC_CIR | Reserved | | | | | | | | CSSC | Reserved | PLLRDYC | HSERDYC | HSIRDYC | LSERDYC | LSIRDYC | Reserved | | | PLLRDYIE | HSERDYIE | HSIRDYIE | LSERDYIE | LSIRDYIE | CSSF | Reserved | | PLLRDYF | HSERDYF | HSIRDYF | LSERDYF | LSIRDYF | |
| | Reset value | | | | | | | | | 0 | | 0 | 0 | 0 | 0 | 0 | | | | 0 | 0 | 0 | 0 | 0 | 0 | | | 0 | 0 | 0 | 0 | 0 | |
| 00Ch | RCC_APB2RSTR | Reserved | | | | | | | | | | | | | | | | USART1RST | Reserved | SPI1RST | TIM1RST | ADC2RST | ADC1RST | Reserved | | IOPERST | IOPDRST | IOPCRST | IOPBRST | IOPARST | Reserved | AFIORST | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | | 0 | 0 | 0 | 0 | | | 0 | 0 | 0 | 0 | 0 | | 0 | |
| 010h | RCC_APB1RSTR | Reserved | | | PWRRST | BKPRST | Reserved | CANRST | Reserved | USBRST | I2C2RST | I2C1RST | Reserved | | USART3RST | USART2RST | Reserved | SPI2RST | Reserved | | WWDGRST | Reserved | | | | | | | | TIM4RST | TIM3RST | TIM2RST | |
| | Reset value | | | | 0 | 0 | | 0 | | 0 | 0 | 0 | | | 0 | 0 | | 0 | | | 0 | | | | | | | | | 0 | 0 | 0 | |
| 014h | RCC_AHBENR | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | FLITFEN | Reserved | SRAMEN | Reserved | DMAEN | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | | 1 | 0 | |
| 018h | RCC_APB2ENR | Reserved | | | | | | | | | | | | | | | | USART1EN | Reserved | SPI1EN | TIM1EN | ADC2EN | ADC1EN | Reserved | | IOPEEN | IOPDEN | IOPCEN | IOPBEN | IOPAEN | Reserved | AFIOEN | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | | 0 | 0 | 0 | 0 | | | 0 | 0 | 0 | 0 | 0 | | 0 | |
| 01Ch | RCC_APB1ENR | Reserved | | | PWREN | BKPEN | Reserved | CANEN | Reserved | USBEN | I2C2EN | I2C1EN | Reserved | | USART3EN | USART2EN | Reserved | SPI2EN | Reserved | | WWDGEN | Reserved | | | | | | | | TIM4EN | TIM3EN | TIM2EN | |
| | Reset value | | | | 0 | 0 | | 0 | | 0 | 0 | 0 | | | 0 | 0 | | 0 | | | 0 | | | | | | | | | 0 | 0 | 0 | |
| 020h | RCC_BDCR | Reserved | | | | | | | | | | | | | | | BDRST | RTCEN | Reserved | | | | RTC SEL [1:0] | | Reserved | | | | | LSEBYP | LSERDY | LSEON | |
| | Reset value | | | | | | | | | | | | | | | | 0 | 0 | | | | | 0 | 0 | | | | | | 0 | 0 | 0 | |
| 024h | RCC_CSR | LPWRSTF | WWDGRSTF | IWDGRSTF | SFTRSTF | PORRSTF | PINRSTF | Reserved | RMVF | Reserved | | | | | | | | | | | | | | | | | | | | | | LSIRDY | LSION |
| | Reset value | 0 | 0 | 0 | 0 | 1 | 1 | | 0 | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 |

RCC 에서 Clock 을 설정하는 가장 중요한 레지스터는 RCC_CR 과 RCC_CFGR 이다. RCC_CR 에서 HSEON 을 SET 하면 HSE 클럭을 사용하는 것이며, RCC_CFGR 에서 PLLSRC 를 HSE 로 설정하고 PLLMUL 에서 클럭의 배수 값을 설정한 후에 SYSCLK 으로 PLLCLK 를 SW 에서 설정한다.

그리고 AHB, APB1, APB2 의 버스에 연결된 Peripheral 들의 Clock Enable 을 담당하는 RCC_APB1ENR, RCC_APB2ENR, RCC_AHBENR 레지스터가 있다.

그 동안 우리가 peripheral 을 사용하기 전에 사용했던 RCC_APBxPeriphClockCmd 함수가 하는 일은 RCC_APBxENR 레지스터의 해당 peripheral Clock 을 Enable 하는 것이다.
해당 Peripheral 에 Clock 이 Enable 되어야 동작을 하기 때문이다.

```
Reset_Handler
        LDR     R0, =SystemInit
        BLX     R0
        LDR     R0, =__iar_program_start
        BX      R0
```

Reset_Handler 에서는 SystemInit 함수를 호출해서 RCC 의 Clock 을 초기화 한다.
클럭 설정이 되어야 내부 시스템을 사용할 수 있다.

### 4.2.5 RCC 초기화 코드

RCC 는 Reset 시 가장 먼저 설정해야 하는 Controller 이기에, SPL 에서는 Reset_Handler 에 SystemInit 함수를 추가해서 Reset 시 RCC 를 초기화 하도록 하고 있다.
만약 SPL 에서 RCC 를 초기화를 해주지 않았다면, RCC 의 기본 Reset 값인 내부 8Mhz 클럭을 Source 로 해서 SYSCLK, PCLK1, PCLK2 등은 모두 8Mhz 로 동작한다.

SystemInit 함수에서 어떻게 RCC 를 초기화 하는지 알아 본다.

1) Reset_Handler

디바이스에 따라 startup_stm32f10x_cl.s, startup_stm32f10x_md.s, startup_stm32f10x_hd.s, 기타 여러가지가 있는데 STM32F103 RCT6 이나, RE6 은 startup_stm32f10x_hd.s 를 사용한다. 여기에 정의되어 있는 Reset_Handler 이다.
Reset 이 걸리면 가장먼저 SystemInit 함수로 jump 하도록 되어 있다.

```
; Reset handler
Reset_Handler    PROC
                 EXPORT  Reset_Handler             [WEAK]
                 IMPORT  __main
                 IMPORT  SystemInit
                 LDR     R0, =SystemInit
                 BLX     R0
                 LDR     R0, =__main
                 BX      R0
                 ENDP
```

2) void SystemInit(void)

이 부분은 프로그램 작성하는 취향에 따라 함수가 다르게 작성될수 있다. 예를 들면 void RCC_Configuration(void)에 있을 수도 있다. 그러나 함수에 들어 있는 내용은 동일하다. SystemInit 함수는 System_stm32f10x.c 파일에 정의되어 있다.

< System_stm32f10x.c>

```
void SystemInit (void)
{
  /* Reset the RCC clock configuration to the default reset state(for debug purpose) */
RCC->CR |= (uint32_t)0x00000001; /* Set HSION bit */

   /* Reset SW, HPRE, PPRE1, PPRE2, ADCPRE and MCO bits */
#ifndef STM32F10X_CL
   RCC->CFGR &= (uint32_t)0xF8FF0000;
#else
   RCC->CFGR &= (uint32_t)0xF0FF0000;
#endif /* STM32F10X_CL */

RCC->CR &= (uint32_t)0xFEF6FFFF; /* Reset HSEON, CSSON and PLLON bits */
RCC->CR &= (uint32_t)0xFFFBFFFF; /* Reset HSEBYP bit */
   /* Reset PLLSRC, PLLXTPRE, PLLMUL and USBPRE/OTGFSPRE bits */
   RCC->CFGR &= (uint32_t)0xFF80FFFF;
#ifdef STM32F10X_CL
RCC->CR &= (uint32_t)0xEBFFFFFF; /* Reset PLL2ON and PLL3ON bits */
RCC->CIR = 0x00FF0000;    /* Disable all interrupts and clear pending bits    */
RCC->CFGR2 = 0x00000000; /* Reset CFGR2 register */

#elif defined (STM32F10X_LD_VL) || defined (STM32F10X_MD_VL) || (defined
STM32F10X_HD_VL)
RCC->CIR = 0x009F0000; /* Disable all interrupts and clear pending bits    */
RCC->CFGR2 = 0x00000000; /* Reset CFGR2 register */

#else
   /* Disable all interrupts and clear pending bits    */
   RCC->CIR = 0x009F0000;
#endif /* STM32F10X_CL */

#if defined (STM32F10X_HD) || (defined STM32F10X_XL) || (defined STM32F10X_HD_VL)
   #ifdef DATA_IN_ExtSRAM
      SystemInit_ExtMemCtl();
   #endif /* DATA_IN_ExtSRAM */
#endif

   /* Configure the System clock frequency, HCLK, PCLK2 and PCLK1 prescalers */
   /* Configure the Flash Latency cycles and enable prefetch buffer */
   SetSysClock();

#ifdef VECT_TAB_SRAM
   SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM.
*/
#else
   SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal
FLASH. */
#endif
}
```

SystemInit 함수가 호출되면 내부 HIS Clock 을 ON 한 다음 HSE, PLL, RCC 인터럽트 설정을 모두 Reset 한다. 그리고 SetSysClock 함수를 호출한다.

3) void SetSysClock(void)

<SetSysClock / System_stm32f10x.c>

```
static void SetSysClock(void)
{
#ifdef SYSCLK_FREQ_HSE
   SetSysClockToHSE();
#elif defined SYSCLK_FREQ_24MHz
   SetSysClockTo24();
#elif defined SYSCLK_FREQ_36MHz
   SetSysClockTo36();
#elif defined SYSCLK_FREQ_48MHz
   SetSysClockTo48();
#elif defined SYSCLK_FREQ_56MHz
   SetSysClockTo56();
#elif defined SYSCLK_FREQ_72MHz
   SetSysClockTo72();
#endif

  /* If none of the define above is enabled, the HSI is used as System clock
      source (default after reset) */
}
```

SetSysClock 함수에서는 Define 된 Clock 설정에 따라서 해당 클럭 함수를 호출한다. 기본값은 SYSCLK_FREQ_72MHz 로 정의되어 있다.

SetSysClockTo72 함수의 72Mhz 클럭 설정 한다.

<SetSysClockTo72 / System_stm32f10x.c>

```
 static void SetSysClockTo72(void)
{
   __IO uint32_t StartUpCounter = 0, HSEStatus = 0;

   /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ----------------------------*/
   /* Enable HSE */
   RCC->CR |= ((uint32_t)RCC_CR_HSEON);

   /* Wait till HSE is ready and if Time out is reached exit */
   do
   {
      HSEStatus = RCC->CR & RCC_CR_HSERDY;
      StartUpCounter++;
   } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

   if ((RCC->CR & RCC_CR_HSERDY) != RESET)
   {       HSEStatus = (uint32_t)0x01;    }
   else
   {       HSEStatus = (uint32_t)0x00;    }

   if (HSEStatus == (uint32_t)0x01)
   {
FLASH->ACR |= FLASH_ACR_PRFTBE; /* Enable Prefetch Buffer */
      /* Flash 2 wait state */
      FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
      FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_2;
```

```
RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1; /* HCLK = SYSCLK */
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV1; /* PCLK2 = HCLK */
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV2; /* PCLK1 = HCLK */
#ifdef STM32F10X_CL
      /* Configure PLLs ------------------------------------------------------*/
      /* PLL2 configuration: PLL2CLK = (HSE / 5) * 8 = 40 MHz */
      /* PREDIV1 configuration: PREDIV1CLK = PLL2 / 5 = 8 MHz */

      RCC->CFGR2 &= (uint32_t)~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2MUL |
                                RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);
      RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL8 |
                               RCC_CFGR2_PREDIV1SRC_PLL2                      |
RCC_CFGR2_PREDIV1_DIV5);

RCC->CR |= RCC_CR_PLL2ON; /* Enable PLL2 */

      /* Wait till PLL2 is ready */
      while((RCC->CR & RCC_CR_PLL2RDY) == 0)
      {       }
/* PLL configuration: PLLCLK = PREDIV1 * 9 = 72 MHz */
      RCC->CFGR      &=      (uint32_t)~(RCC_CFGR_PLLXTPRE      |      RCC_CFGR_PLLSRC      |
RCC_CFGR_PLLMULL);
      RCC->CFGR                |=                (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1                |
RCC_CFGR_PLLSRC_PREDIV1 |
                                    RCC_CFGR_PLLMULL9);
#else
      /*    PLL configuration: PLLCLK = HSE * 9 = 72 MHz */
      RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_PLLSRC | RCC_CFGR_PLLXTPRE |
                                          RCC_CFGR_PLLMULL));
      RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLSRC_HSE | RCC_CFGR_PLLMULL9);
#endif /* STM32F10X_CL */
RCC->CR |= RCC_CR_PLLON; /* Enable PLL */
while((RCC->CR & RCC_CR_PLLRDY) == 0) /* Wait till PLL is ready */
      {       }

RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW)); /* Select PLL as system clock source */
      RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
      /* Wait till PLL is used as system clock source */
      while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08)
      {       }
   }
   else
   { /* If HSE fails to start-up, the application will have wrong clock
            configuration. User can add here some code to deal with this error */
   }
}
```

RCC_CR_HSEON 을 Set 하여 외부 클럭을 ON 하고, HSE Clock 이 정상적으로 준비가 되었는지 확인하기 위해서 RCC_CR_HSERDY 을 체크한다.
HSE 클럭이 준비가 된 후 PLL 을 설정하기 전 내부 버스들의 prescaler 값을 설정한다.
HCLK, PCLK2 은 DIV1 으로 설정하여 SYSCLK 와 동일한 72Mhz 가 되도록 하고, PCLK1 은 DIV2 하여 36Mhz 가 되도록 한다. 다음으로 PLLCLK 를 HSE 의 9 배가 되도록 RCC->CFGR 의 PLLSRC 와 PLLMUL 을 설정한다.

PLL 설정이 완료되었으면 PLLON 을 하고, PLL 이 준비되었는지 RCC_CR_PLLRDY 을 체크한다. PLL 이 준비가 되었으면 RCC->CFGR 의 SW 를 PLLCLK 로 선택한다. SW 가 정상적으로 PLLCLK 로 설정이 되었는지 RCC->CFGR 의 SWS 를 체크한다.

4) Clock 설정 확인

RCC_GetClocksFreq 함수는 RCC_GetClocksFreq 구조체의 정보를 return 한다. SYSCLK, HCLK, PCLK1, PCLK2, ADCCLK 값을 읽을 수 있다.

<RCC_ClocksTypeDef>

```
typedef struct
{
  uint32_t SYSCLK_Frequency;   /*!< returns SYSCLK clock frequency expressed in Hz */
  uint32_t HCLK_Frequency;     /*!< returns HCLK clock frequency expressed in Hz */
  uint32_t PCLK1_Frequency;    /*!< returns PCLK1 clock frequency expressed in Hz */
  uint32_t PCLK2_Frequency;    /*!< returns PCLK2 clock frequency expressed in Hz */
  uint32_t ADCCLK_Frequency;   /*!< returns ADCCLK clock frequency expressed in Hz */
}RCC_ClocksTypeDef;
```

RCC_GetSYSCLKSource 함수는 SYSCLK 의 Clock Source 가 어떤 것인지를 return 한다. SYSCLK 의 Clock Source 값은 RCC->CFGR 의 SWS 값으로 읽을 수 있으며 Return 값은 다음과 같다.

```
0x00: HSI used as system clock
0x04: HSE used as system clock
0x08: PLL used as system clock
```

RCC_GetSYSCLKSource, RCC_GetClocksFreq 함수를 사용하면 현재 시스템 Clock 정보를 읽어 올 수 있다.

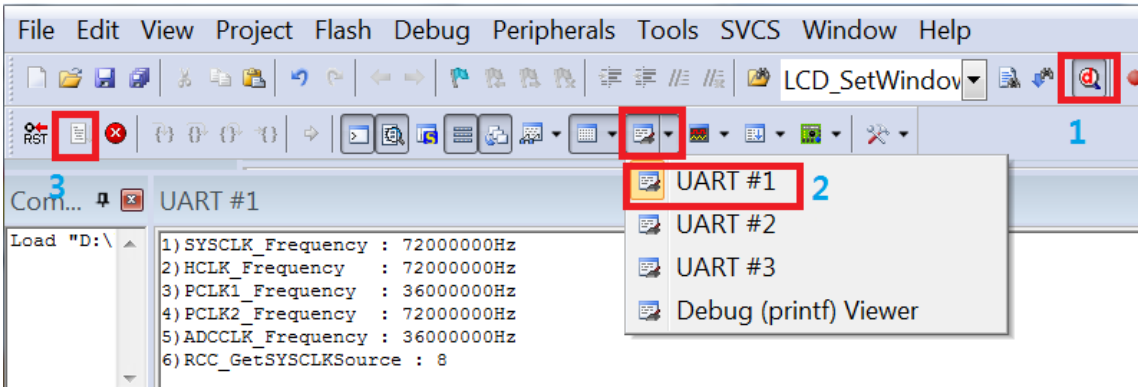
```
#include "stm32f10x.h"
#include <stdio.h>

size_t __write(int handle, const unsigned char *buf, size_t bufSize)
{
        size_t nChars = 0;
        if(handle == -1) return 0;
        else if(handle != 1 && handle != 2) return -1;

        for(;bufSize> 0; --bufSize)
        {
                USART_SendData(USART1, (uint8_t)*buf);
                /* Loop until the end of transmission */
                while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET)
                {}
              ++buf;
```

```
                ++nChars;
        }

   return nChars;
}

int main()
{
        GPIO_InitTypeDef GPIO_InitStructure;
        USART_InitTypeDef USART_InitStructure;
        RCC_ClocksTypeDef RCC_ClockFreq;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_Init(GPIOA, &GPIO_InitStructure);

        USART_InitStructure.USART_BaudRate = 115200;
        USART_InitStructure.USART_WordLength = USART_WordLength_8b;
        USART_InitStructure.USART_StopBits = USART_StopBits_1;
        USART_InitStructure.USART_Parity = USART_Parity_Even;
        USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
        USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

        USART_Init(USART1, &USART_InitStructure);
        USART_Cmd(USART1, ENABLE);

        RCC_GetClocksFreq(&RCC_ClockFreq);

        printf("1)SYSCLK_Frequency : %d\r\n", RCC_ClockFreq.SYSCLK_Frequency);
        printf("2)HCLK_Frequency : %d\r\n", RCC_ClockFreq.HCLK_Frequency);
        printf("3)PCLK1_Frequency : %d\r\n", RCC_ClockFreq.PCLK1_Frequency);
        printf("4)PCLK2_Frequency: %d\r\n", RCC_ClockFreq.PCLK2_Frequency);
        printf("5)ADCCLK_Frequency : %d\r\n", RCC_ClockFreq.ADCCLK_Frequency);
        printf("6)RCC_GetSYSCLKSource : %x\r\n", RCC_GetSYSCLKSource());

        while(1);
}
```

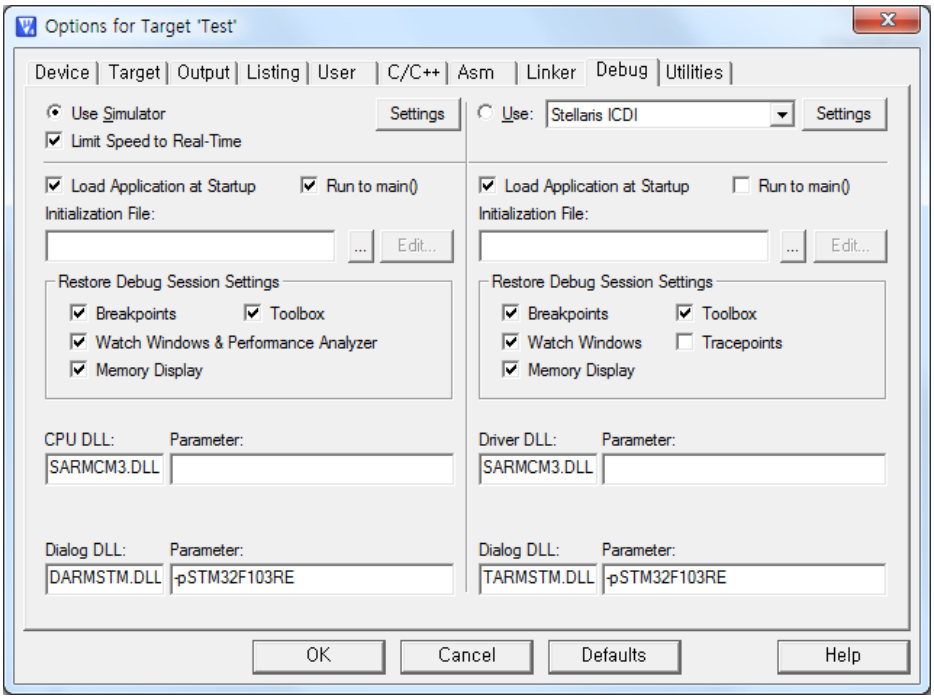
<결과 화면>

```
1)SYSCLK_Frequency : 72000000
2)HCLK_Frequency : 72000000
3)PCLK1_Frequency : 36000000
4)PCLK2_Frequency: 72000000
5)ADCCLK_Frequency : 36000000
6)RCC_GetSYSCLKSource : 8
```

예제 프로그램 4_RCC_UASRT 를 실행하면 위와 같은 결과를 얻을 수 있다.

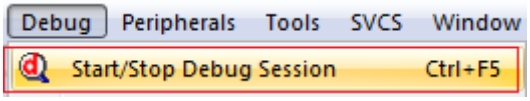


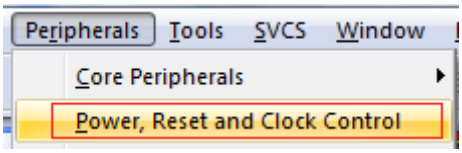

참고로 µvision 는 위와 같은 과정을 거치지 않고 바로 알 수 있는 방법이 있다.
먼저 옵션에서 Use Simulator 를 선택한다.



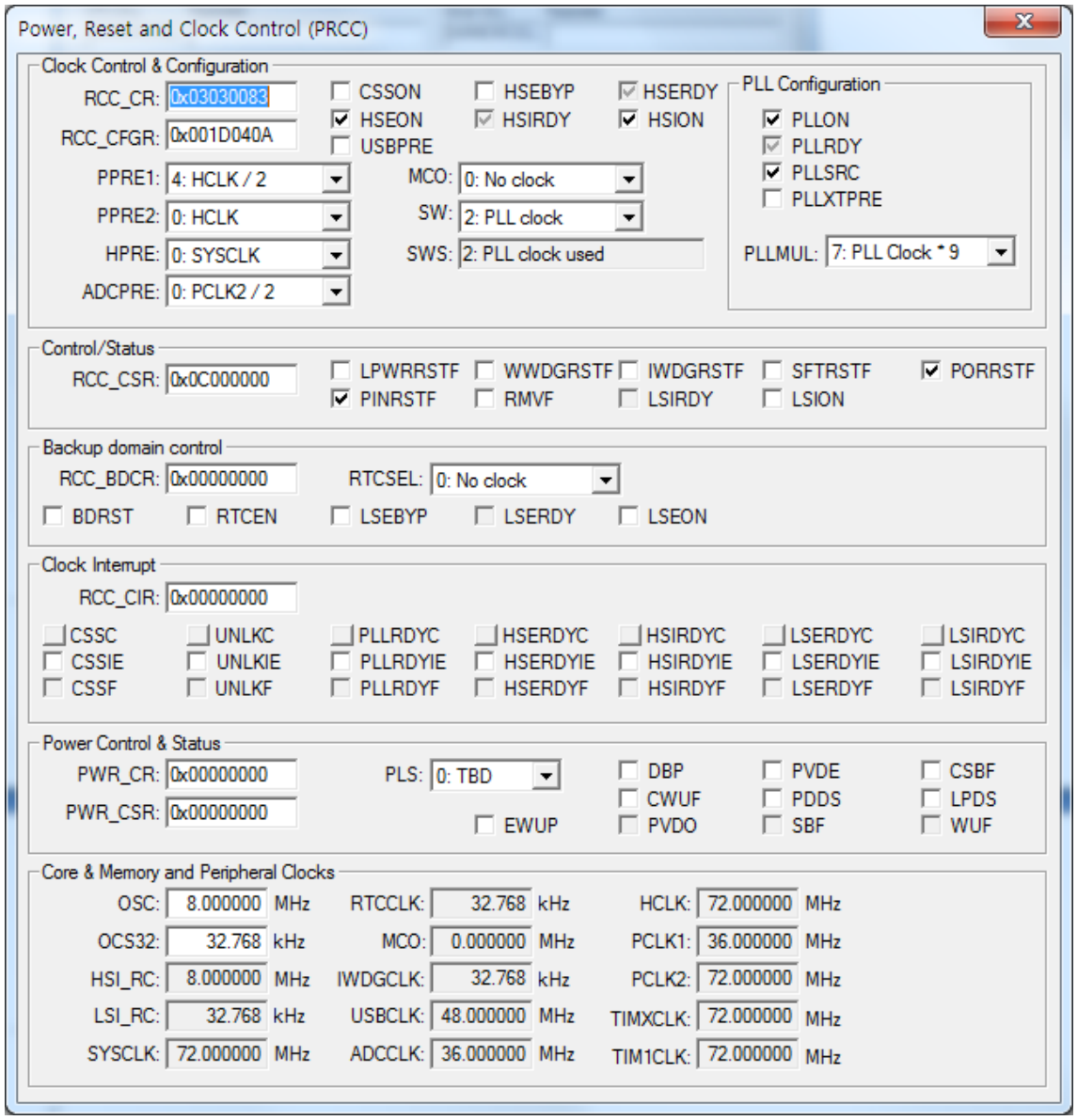

옵션에서 Use Simulator 를 선택하면 개발하는데, 결과를 예측하는데 도움이 된다.
다른 개발 툴에 없는 장점이다.

그리고 다음그림을 클릭하는 과정을 거친다.






다음과 같은 결과를 볼 수 있다.

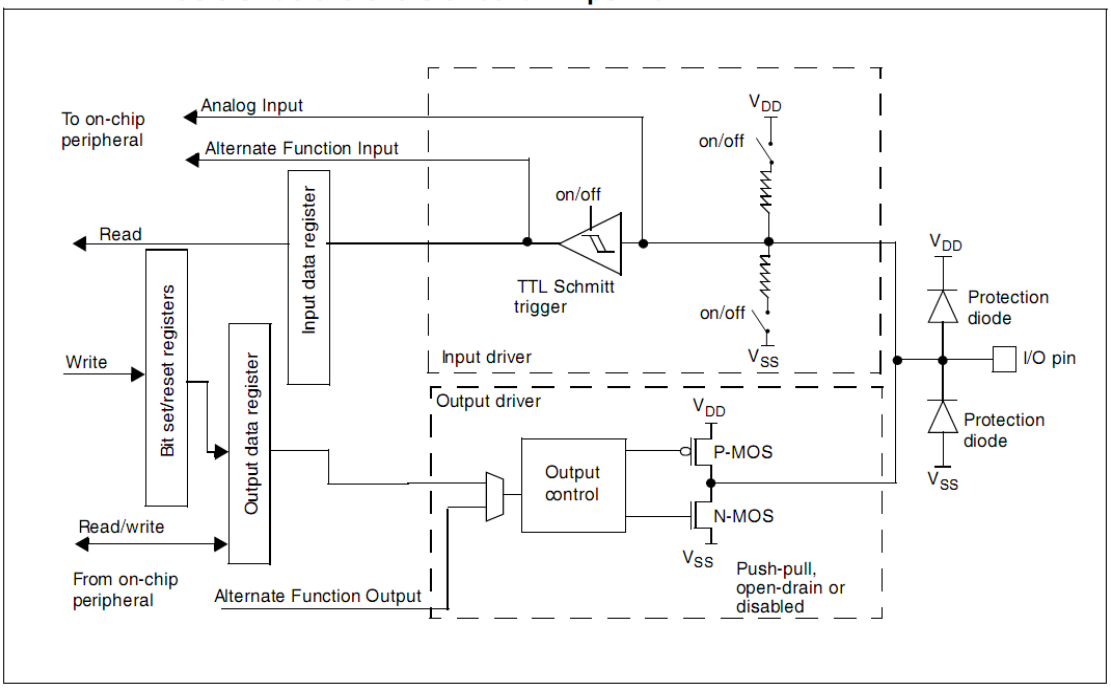


## 4.3 GPIO

### 4.3.1 GPIO 기본구조

GPIO 는 General Purpose Input/Output 약자이며, 따라서 Input, Output으로도 쓸 수 있고, 또는 다른 기능으로도 사용한다.
아래 [그림 4.1]은 STM32 의 GPIO 구조이다. 위의 점선부분은 입력 부이며, 아래 점선부분은 출력 부이다. 그림 오른쪽 I/O pin 이 실제 사용되는 핀으로 Protection diode 는 외부 노이즈로부터 I/O pin 을 보호하기 위해서이다. TTL 슈미트 입력 단에 풀 업, 풀다운 스위치는 입력 신호를 풀 업, 풀다운 기능을 설정할 수 있다. 이곳에서는 AD 변환을 이용할 수 있다.
Output Control 에 P-MOS, N-MOS 회로는 출력 상태에서 Push-pull 또는 Open-drain 을 설정할 수 있다.

**Basic structure of a standard I/O port bit**




[그림 4.1] STM32 의 GPIO 구조

1) Analog input : A/D 변환한 아날로그 값을 읽을 수 있다.
2) Alternate Function input : 슈미트 트리거를 통해 H/L 가 선택된다.
3) Read : Input Data Register 를 통해서 Read 한다.
   Input Data Register 를 통해 사용자는 해당 핀의 값을 읽을 수 있다.
4) Write : Bit Set/Reset register 가 바로 Output data Register 로 되어 있다.
   즉, Bit Set/Reset Register 에 값을 쓰면, Output Data Register 를 통해서 출력 값이 설정된다.

5) Read/Write :Output Data Register 의 값을 읽고 쓸 수 있다.
사용자는 이 선을 통해서 현재의 출력 값을 변경하고, 출력 값을 알 수 있다.
6) Alternate Function Output : Alternate Function 에서 출력을 설정하는 선이다.

### 4.3.2 GPIO Register Map

### 4.3.3

STM32F103x 의 GPIO 는 APB2 버스와 연결되어 있으며, A,B,C,D,E,F,G 7 개의 GPIO 포트를 가지고 있고, 각 포트는 최대 16 개의 핀을 가지고 있다. 여기서 STM32F103 은 PORT 는 Port A,B,C 만 사용하고, PORT D,E,F,G 는 다른 디바이스에서 사용한다.

**Register boundary addresses**

| Boundary address | Peripheral | Bus | Register map |
|---|---|---|---|
| 0x4001 5800 - 0x4001 7FFF | Reserved | | |
| 0x4001 5400 - 0x4001 57FF | TIM11 timer | | *Section 16.6.11 on page 447* |
| 0x4001 5000 - 0x4001 53FF | TIM10 timer | | *Section 16.6.11 on page 447* |
| 0x4001 4C00 - 0x4001 4FFF | TIM9 timer | | *Section 16.5.13 on page 438* |
| 0x4001 4000 - 0x4001 4BFF | Reserved | | |
| 0x4001 3C00 - 0x4001 3FFF | ADC3 | | *Section 11.12.15 on page 243* |
| 0x4001 3800 - 0x4001 3BFF | USART1 | | *Section 27.6.8 on page 797* |
| 0x4001 3400 - 0x4001 37FF | TIM8 timer | | *Section 14.4.21 on page 347* |
| 0x4001 3000 - 0x4001 33FF | SPI1 | | *Section 25.5 on page 712* |
| 0x4001 2C00 - 0x4001 2FFF | TIM1 timer | | *Section 14.4.21 on page 347* |
| 0x4001 2800 - 0x4001 2BFF | ADC2 | APB2 | *Section 11.12.15 on page 243* |
| 0x4001 2400 - 0x4001 27FF | ADC1 | | *Section 11.12.15 on page 243* |
| 0x4001 2000 - 0x4001 23FF | GPIO Port G | | *Section 9.5 on page 188* |
| 0x4001 1C00 - 0x4001 1FFF | GPIO Port F | | *Section 9.5 on page 188* |
| 0x4001 1800 - 0x4001 1BFF | GPIO Port E | | *Section 9.5 on page 188* |
| 0x4001 1400 - 0x4001 17FF | GPIO Port D | | *Section 9.5 on page 188* |
| 0x4001 1000 - 0x4001 13FF | GPIO Port C | | *Section 9.5 on page 188* |
| 0x4001 0C00 - 0x4001 0FFF | GPIO Port B | | *Section 9.5 on page 188* |
| 0x4001 0800 - 0x4001 0BFF | GPIO Port A | | *Section 9.5 on page 188* |
| 0x4001 0400 - 0x4001 07FF | EXTI | | *Section 10.3.7 on page 205* |
| 0x4001 0000 - 0x4001 03FF | AFIO | | *Section 9.5 on page 188* |

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | **GPIOx_CRL** | CNF7 [1:0] | | MODE7 [1:0] | | CNF6 [1:0] | | MODE6 [1:0] | | CNF5 [1:0] | | MODE5 [1:0] | | CNF4 [1:0] | | MODE4 [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] | |
| | Reset value | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0x04 | **GPIOx_CRH** | CNF 15 [1:0] | | MODE15 [1:0] | | CNF 14 [1:0] | | MODE14 [1:0] | | CNF 13 [1:0] | | MODE13 [1:0] | | CNF 12 [1:0] | | MODE12 [1:0] | | CNF 11 [1:0] | | MODE11 [1:0] | | CNF 10 [1:0] | | MODE10 [1:0] | | CNF 9 [1:0] | | MODE9 [1:0] | | CNF 8 [1:0] | | MODE8 [1:0] | |
| | Reset value | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0x08 | **GPIOx_IDR** | Reserved | | | | | | | | | | | | | | | | IDR[15:0]I | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x0C | **GPIOx_ODR** | Reserved | | | | | | | | | | | | | | | | ODR[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x10 | **GPIOx_BSRR** | BR[15:0] | | | | | | | | | | | | | | | | BSR[15:0] | | | | | | | | | | | | | | | |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x14 | **GPIOx_BRR** | Reserved | | | | | | | | | | | | | | | | BR[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x18 | **GPIOx_LCKR** | Reserved | | | | | | | | | | | | | | | LCKK | LCK[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

[Table 4.1] GPIO Register APB2 Base Address

| | |
|---|---|
| GPIOx_CRL | 0~7 핀의 GPIO 설정을 한다. |
| GPIOx_CRH | 8~15 핀의 GPIO 설정을 한다. |
| GPIOx_IDR | 핀의 입력 상태를 읽는다. |
| GPIOx_ODR | 핀의 출력 값을 설정하고, 출력 상태를 읽는다. |
| GPIOx_BSRR | 상위 BR 파트는 핀을 Reset 한다. 하위 BSR 파트는 핀을 Set 한다.<br>만약 BR 과 BSR 이 동일 핀을 같이 set 하는 경우 BSR 의 값이 된다. |
| GPIOx_BRR | 판을 Reset 하는 기능을 한다. BSRR 의 BR 파트와 동일 기능을 하는 레지스터이다. |
| GPIOx_LCKR | LCKR 이 Set 된 핀은 Control Register 의 설정 변경이 불가능해 진다.<br>한번 Set 을 하면 Device 를 Reset 해야만 Control Register 의 재설정이 가능하다. |

[Table 4.2] GPIO Register

[Table 4.1]와 [Table 4.2]을 보면 각 포트는 자신의 Base Address 에 다음 GPIO Register 들을 모두 가지고 있다. 예를 들면, GPIOA_ODR 레지스터의 주소는 0x40010800+0x0c 이며 , GPIOB_ODR 레지스터의 주소는 0x40010C00+0x0c 이며, GPIOC_ODR 레지스터의 주소는 0x40011000+0x0c 가 이다.

-Reset and clock control register (RCC)
[Table 4.3] RCC register map 이며 리셋시 값을 나타낸다.

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000h | **RCC_CR** | Reserved | | | | | | PLL RDY | PLL ON | Reserved | | | | CSSON | HSEBYP | HSERDY | HSEON | HSICAL[7:0] | | | | | | | | HSITRIM[4:0] | | | | | Reserved | HSIRDY | HSION |
| | Reset value | | | | | | | 0 | 0 | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | | 1 | 1 |
| 004h | **RCC_CFGR** | Reserved | | | | MCO [2:0] | | | | Reserved | USBPRE | PLLMUL[3:0] | | | | PLLXTPRE | PLLSRC | ADC PRE [1:0] | | PPRE2 [2:0] | | | PPRE1 [2:0] | | | HPRE[3:0] | | | | SWS [1:0] | | SW [1:0] | |
| | Reset value | | | | | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 008h | **RCC_CIR** | Reserved | | | | | | | | CSSC | Reserved | | PLLRDYC | HSERDYC | HSIRDYC | LSERDYC | LSIRDYC | Reserved | | | PLLRDYIE | HSERDYIE | HSIRDYIE | LSERDYIE | LSIRDYIE | CSSF | Reserved | | PLLRDYF | HSERDYF | HSIRDYF | LSERDYF | LSIRDYF |
| | Reset value | | | | | | | | | 0 | | | 0 | 0 | 0 | 0 | 0 | | | | 0 | 0 | 0 | 0 | 0 | 0 | | | 0 | 0 | 0 | 0 | 0 |
| 00Ch | **RCC_APB2RSTR** | Reserved | | | | | | | | | | | | | | | | | USART1RST | Reserved | SPI1RST | TIM1RST | ADC2RST | ADC1RST | Reserved | IOPERST | IOPDRST | IOPCRST | IOPBRST | IOPARST | Reserved | AFIORST |
| | Reset value | | | | | | | | | | | | | | | | | | 0 | | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | | 0 |
| 010h | **RCC_APB1RSTR** | Reserved | | | PWRRST | BKPRST | Reserved | CANRST | Reserved | USBRST | I2C2RST | I2C1RST | Reserved | | USART3RST | USART2RST | Reserved | | SPI2RST | Reserved | | WWDGRST | Reserved | | | | | | | | TIM4RST | TIM3RST | TIM2RST |
| | Reset value | | | | 0 | 0 | | 0 | | 0 | 0 | 0 | | | 0 | 0 | | | 0 | | | 0 | | | | | | | | 0 | 0 | 0 |
| 014h | **RCC_AHBENR** | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | | 1 | | 1 | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | | | FLITFEN | Reserved | SRAMEN | Reserved | DMAEN | |
| 018h | **RCC_APB2ENR** | Reserved | | | | | | | | | | | | | | | | | USART1EN | Reserved | SPI1EN | TIM1EN | ADC2EN | ADC1EN | Reserved | IOPEEN | IOPDEN | IOPCEN | IOPBEN | IOPAEN | Reserved | AFIOEN |
| | Reset value | | | | | | | | | | | | | | | | | | 0 | | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | | 0 |
| 01Ch | **RCC_APB1ENR** | Reserved | | | PWREN | BKPEN | Reserved | CANEN | Reserved | USBEN | I2C2EN | I2C1EN | Reserved | | USART3EN | USART2EN | Reserved | | SPI2EN | Reserved | | WWDGEN | Reserved | | | | | | | | TIM4EN | TIM3EN | TIM2EN |
| | Reset value | | | | 0 | 0 | | 0 | | 0 | 0 | 0 | | | 0 | 0 | | | 0 | | | 0 | | | | | | | | 0 | 0 | 0 |
| 020h | **RCC_BDCR** | Reserved | | | | | | | | | | | | | | | BDRST | RTCEN | Reserved | | | | | RTC SEL [1:0] | | Reserved | | | | | LSEBYP | LSERDY | LSEON |
| | Reset value | | | | | | | | | | | | | | | | 0 | 0 | | | | | | 0 | 0 | | | | | | 0 | 0 | 0 |
| 024h | **RCC_CSR** | LPWRSTF | WWDGRSTF | IWDGRSTF | SFTRSTF | PORRSTF | PINRSTF | Reserved | RMVF | Reserved | | | | | | | | | | | | | | | | | | | | | LSIRDY | LSION |
| | Reset value | 0 | 0 | 0 | 0 | 1 | 1 | | 0 | | | | | | | | | | | | | | | | | | | | | | 0 | 0 |

[Table 4.3] RCC register map 과 reset 값

RCC_CR : Clock Control
RCC_CFGR : Clock Configuration
RCC_CIR : Clock Interrupt
RCC_APB2RSTR : APB2 Peripheral Reset
RCC_APB1RSTR : APB1 Peripheral Reset
RCC_AHBENR : AHB Peripheral Clock Enable
RCC_APB2ENR : APB2 Peripheral Clock Enable
RCC_APB1ENR : APB1 Peripheral Clock Enable
RCC_BDCR : Backup Domain Control
RCC_CSR : Control/Status

아래 표를 보면 각 핀의 출력, 입력, 설정을 바꿀 수 있다. Output Mode Bits 에서는 GPIO 의 Speed 를 설정이 가능하다.

**Port bit configuration table**

| Configuration mode | | CNF1 | CNF0 | MODE1 | MODE0 | PxODR register |
|---|---|---|---|---|---|---|
| General purpose output | Push-pull | 0 | 0 | 01 | | 0 or 1 |
| | Open-drain | | 1 | 10 | | 0 or 1 |
| Alternate Function output | Push-pull | 1 | 0 | 11 | | don't care |
| | Open-drain | | 1 | see *Table 19* | | don't care |
| Input | Analog | 0 | 0 | 00 | | don't care |
| | Input floating | | 1 | | | don't care |
| | Input pull-down | 1 | 0 | | | 0 |
| | Input pull-up | | | | | 1 |

**Output MODE bits**

| MODE[1:0] | Meaning |
|---|---|
| 00 | Reserved |
| 01 | Max. output speed 10 MHz |
| 10 | Max. output speed 2 MHz |
| 11 | Max. output speed 50 MHz |

다음은 IO 지정방법이다.

A. 상대 주소 지정 방식

```
RCC->APB2ENR|=1<<3;       //PORT 클럭 활성화
GPIOB->CRH&=0XFFFFFFF0;
GPIOB->CRH|=0X00000003;//PB   8~15핀의 GPIO
GPIOB->ODR|=1<<8;          //PB8   GPIO Mode =>Output
```

B. 라이브러리 사용방식

```
GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOB , ENABLE);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;                  // Port      PB8
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//    GPIO Speed 50MHz
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // GPIO Mode =>Output
GPIO_Init(GPIOB, &GPIO_InitStructure);                     // Port B
```

테스트 프로그램에서는 예 1 보다 예 2 를 사용 하였다.

### 4.3.4 GPIO 입출력 함수

1) GPIO 출력 설정 함수

```
void    GPIO_Init (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)   //초기화
void    GPIO_DeInit (GPIO_TypeDef *GPIOx)                                     //Reset 기본 초기화
void    GPIO_SetBits (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)                 //High 로 설정
void    GPIO_ResetBits (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)               //Low 로 설정
uint16_t    GPIO_ReadOutputData (GPIO_TypeDef *GPIOx)                         //출력  읽기
uint8_t    GPIO_ReadOutputDataBit (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)    //포트 출력 읽기
void    GPIO_PinLockConfig (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)           //포트 Lock 설정
void    GPIO_Write (GPIO_TypeDef *GPIOx, uint16_t PortVal)                    //ODR 레지스터 쓰기
```

2) GPIO 입력 설정 함수

```
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)     //포트입력 값을 읽기

uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx)                          //IDR 레지스터 읽기
```

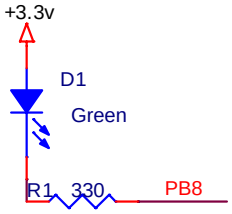
3) 예 제

포트를 1(High)과 0(Low) 을 출력하기 위해서는 stm32f10x_gpio.c 내용중
아래 두 함수를 이용한다.
1(High)은 void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) 를
0(Low)은 void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) 를 사용한다.

```
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
assert_param(IS_GPIO_PIN(GPIO_Pin));
GPIOx->BSRR = GPIO_Pin;
}
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
assert_param(IS_GPIO_PIN(GPIO_Pin));
GPIOx->BRR = GPIO_Pin;
}
```

PB8 포트를 1 로 할 때 GPIO_SetBits(GPIOB , GPIO_Pin_8); 사용하고
PB8 포트를 0 으로 할 때 GPIO_ResetBits(GPIOB , GPIO_Pin_8); 사용한다.

D1이 PB8 에 연결이 되어 있다. Low 일 때 LED 가
ON 되고, Low 일 때 High 이면 OFF된다.




[그림 4.2] 회로

1_GPIO_LED 의 main 함수이다.

예 1)

```
int main(void)
{           SystemInit();
            GPIO_Configuration();
      /* Infinite loop */
      while (1){          /*====D1-OFF=======*/
                          GPIO_SetBits(GPIOB , GPIO_Pin_8);
                          Delay(0xfffff);
                          /*====D1-ON=======*/
                          GPIO_ResetBits(GPIOB , GPIO_Pin_8);
                          Delay(0xfffff);
      }
}
```

예 2)

```
#define LED     GPIO_Pin_8
#define LED_H    GPIOB->BSRR =LED
#define LED_L    GPIOB->BRR    =LED
while (1){
                          LED_H; /*====D1-OFF=======*/
                          Delay(0xfffff);
                          LED_L; /*====D1-ON=======*/
                          Delay(0xfffff);
      }
}
```

예 1)과 예 2) 같은 내용이나 예 2) 는 LED_H 와 LED_L 는 자주 사용할 때 이런 식으로 한다. 자세한 결과 내용은 실습편 GPIO_LED 을 보자.

<입력포트>

I/O 포트를 입력으로 할때는 GPIO_InitStructure.GPIO_Mode =GPIO_Mode_IPU; 처럼 GPIO_Mode_IPU 를 사용한다.

예제 프로그램 4_USART 2 를 참고한다.

### 4.4 SysTick

Delay 함수보다 좀더 정확하게 사용하는 경우 사용한다.

일반적으로 사용할 때

```
void    Delay (uint32_t nCount)
{
  for(; nCount != 0; nCount--);
}

void delay_ms(uint16_t nms)
{
      uint32_t temp = delay_fac_ms * nms;
      if (temp > 0x00ffffff)
      {
            temp = 0x00ffffff;
      }
      SysTick_SetReload(temp);
      SysTick_CounterCmd(SysTick_Counter_Clear);
      SysTick_CounterCmd(SysTick_Counter_Enable);
      do
      {
            Status = SysTick_GetFlagStatus(SysTick_FLAG_COUNT);
      }while (Status != SET);
      SysTick_CounterCmd(SysTick_Counter_Disable);
            SysTick_CounterCmd(SysTick_Counter_Clear);
}

void delay_us(u32 Nus)
{
      SysTick_SetReload(delay_fac_us * Nus);
      SysTick_CounterCmd(SysTick_Counter_Clear);
      SysTick_CounterCmd(SysTick_Counter_Enable);
      do
      {
            Status = SysTick_GetFlagStatus(SysTick_FLAG_COUNT);
      }while (Status != SET);
      SysTick_CounterCmd(SysTick_Counter_Disable);
            SysTick_CounterCmd(SysTick_Counter_Clear);
}
```

위의 내용을 가지고 사용

```
            delay_ms(1000);     /* delay 1000ms */
```

초기에 Reset 과 Clock Control 관련 레지스터를 초기화하고 셋팅 한 후 사용할 클럭들을 Enable 한다. Enable 하기 위한 함수는 RCC_APB2PeriphClockCmd(); 아래에는 이 함수를 나타낸다.

stm32f10x_rcc.c

```
void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState)
{
  /* Check the parameters */
  assert_param(IS_RCC_APB2_PERIPH(RCC_APB2Periph));
  assert_param(IS_FUNCTIONAL_STATE(NewState));
  if (NewState != DISABLE)
  {
    RCC->APB2ENR |= RCC_APB2Periph;
  }
  else
  {
    RCC->APB2ENR &= ~RCC_APB2Periph;
  }
}
```

system_stm32f10x.c

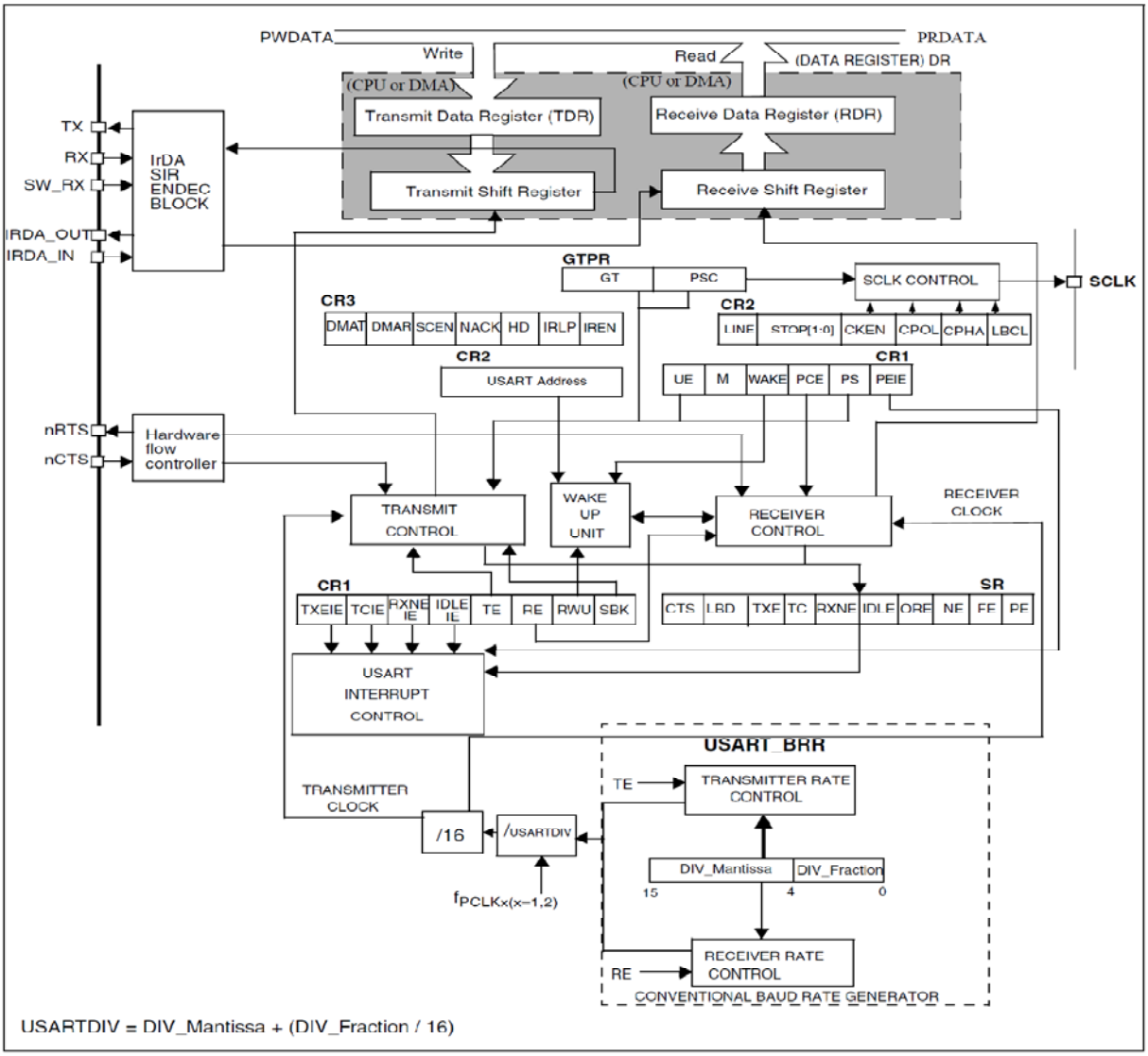
```
void SystemInit (void)
{
  /* Reset the RCC clock configuration to the default reset state(for debug purpose) */
  /* Set HSION bit */
  RCC->CR |= (uint32_t)0x00000001;

  /* Reset SW, HPRE, PPRE1, PPRE2, ADCPRE and MCO bits */
#ifndef STM32F10X_CL
  RCC->CFGR &= (uint32_t)0xF8FF0000;
#else
  RCC->CFGR &= (uint32_t)0xF0FF0000;
#endif /* STM32F10X_CL */

  /* Reset HSEON, CSSON and PLLON bits */
  RCC->CR &= (uint32_t)0xFEF6FFFF;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;

  /* Reset PLLSRC, PLLXTPRE, PLLMUL and USBPRE/OTGFSPRE bits */
  RCC->CFGR &= (uint32_t)0xFF80FFFF;

#ifdef STM32F10X_CL
  /* Reset PLL2ON and PLL3ON bits */
  RCC->CR &= (uint32_t)0xEBFFFFFF;

  /* Disable all interrupts and clear pending bits  */
  RCC->CIR = 0x00FF0000;

  /* Reset CFGR2 register */
  RCC->CFGR2 = 0x00000000;
#elif defined (STM32F10X_LD_VL) || defined (STM32F10X_MD_VL) || (defined
STM32F10X_HD_VL)
  /* Disable all interrupts and clear pending bits  */
  RCC->CIR = 0x009F0000;
```

```
  /* Reset CFGR2 register */
  RCC->CFGR2 = 0x00000000;
#else
  /* Disable all interrupts and clear pending bits  */
  RCC->CIR = 0x009F0000;
#endif /* STM32F10X_CL */

#if defined (STM32F10X_HD) || (defined STM32F10X_XL) || (defined STM32F10X_HD_VL)
  #ifdef DATA_IN_ExtSRAM
    SystemInit_ExtMemCtl();
  #endif /* DATA_IN_ExtSRAM */
#endif

  /* Configure the System clock frequency, HCLK, PCLK2 and PCLK1 prescalers */
  /* Configure the Flash Latency cycles and enable prefetch buffer */
  SetSysClock();

#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM.
*/
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal
FLASH. */
#endif
}
```

## 4.5 USART

### 4.5.1 USART 구조

Uiversal Synchronous Asynchronous Receiver Transmitter (USART)의 약자로 범용 동기 비동기 송수신기이다. NRZ 의 비동기 직렬 데이터 형식이고, 외부 장비와 풀 - 듀플렉스 데이터 교환 뿐 아니라, 고속 데이터 통신 Multibuffer 구성에 대해 DMA 를 사용 가능하다.
STM32 의 USART Controller 는 다음 표와 같이 9 개의 모드를 지원한다. USART1~5 까지 총 5 개의 포트가 표시되어 있으나, USART 수는 device 마다 다르다.
**여기서 사용한 STM32F103 보드에서는 2 개를 지원한다.**
USART 인터페이스 내부 구조는 아래 그림처럼 구성 되어 있다. USART 양방향 통신 (Asynchronous Mode)은 데이터 (RX)를 수신과 데이터 출력 (TX)를 전송 한다.




[그림 4.5.1] USART 구조

[그림 4.5.1] 위에 Transmit Data Register (TDR)은 Data 를 전송 할 때 사용하는 레지스터 이고, Receive Data register (RDR)은 수신된 Data 를 읽을 때 사용하는 레지스터 이다. TDR 과 RDR 은 내부적으로 분리되어 있으나 DR(Data Register)이라는 하나의 레지스터이다.
즉, DR 에 데이터를 Write 하면 TDR 로 연결되고, Read 하면 RDR 로 연결되는 구조이다.
TDR 에 데이터를 Write 하면 TSR(Transmit Shift Register)에 의해서 해당 데이터가 TX 로 전송되게 이다. TSR 은 TRANSMIT CONTROL 에서 제어되고 있다. TRANSMIT CONTROL 은 PCLK 에서 USARTDIV 로 분주된 Clock 신호를 받는다. 이 클럭은 Baudrate 를 의미한다.
CR1 의 TE 비트는 Transmit Enable 을 의미하며, TRANSMIT CONTROL 의 상태는 SR 레지스터에 전달한다.
SR 레지스터는 USART 의 상태를 나타내는 Status 레지스터로 SR 의 상태가 USART INTERRUPT CONTROL 로 전달한다. USART INTERRUPT CONTROL 는 SR 에서 들어오는 상태 정보에 따라서 CR1 에 Enable 된 인터럽트들을 발생시킨다.
USART_BRR 은 USART 로 들어오는 PCLK 클럭의 분주비를 설정하는 레지스터이다.

<USART Register>

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | **USART_SR** | Reserved | | | | | | | | | | | | | | | | | | | | | | CTS | LBD | TXE | TC | RXNE | IDLE | ORE | NE | FE | PE |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x04 | **USART_DR** | Reserved | | | | | | | | | | | | | | | | | | | | | | DR[8:0] | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x08 | **USART_BRR** | Reserved | | | | | | | | | | | | | | | | DIV_Mantissa[15:4] | | | | | | | | | | | | DIV_Fraction [3:0] | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x0C | **USART_CR1** | Reserved | | | | | | | | | | | | | | | | | | UE | M | WAKE | PCE | PS | PEIE | TXEIE | TCIE | RXNEIE | IDLEIE | TE | RE | RWU | SBK |
| | Reset value | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x10 | **USART_CR2** | Reserved | | | | | | | | | | | | | | | | | LINEN | STOP [1:0] | | CLKEN | CPOL | CPHA | LBCL | Reserved | LBDIE | LBDL | Reserved | ADD[3:0] | | | |
| | Reset value | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x14 | **USART_CR3** | Reserved | | | | | | | | | | | | | | | | | | | | | CTSIE | CTSE | RTSE | DMAT | DMAR | SCEN | NACK | HDSEL | IRLP | IREN | EIE |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x18 | **USART_GTPR** | Reserved | | | | | | | | | | | | | | | | GT[7:0] | | | | | | | | PSC[7:0] | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

그림 USART Register Map

### 4.5.2 프로그램 기본 구조.

```
/* USART_InitStruct members default value */
   USART_InitStruct->USART_BaudRate = 115200;   // 전송속도 115200 인 경우
   USART_InitStruct->USART_WordLength = USART_WordLength_8b;
   USART_InitStruct->USART_StopBits = USART_StopBits_1;
   USART_InitStruct->USART_Parity = USART_Parity_No ;
   USART_InitStruct->USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
   USART_InitStruct->USART_HardwareFlowControl = USART_HardwareFlowControl_None;
```

1) USART Baudrate

USART 의 Baudrate 는 USART_BRR 레지스터에서 설정한다. Baudrate 는 BRR 레지스터의 DIV_Mantissa 와 DIV_Fraction 에 의해서 결정되며, 계산 공식은 다음과 같다.

USARTDIV = DIV_Mantissa + (DIV_Fraction / 16)

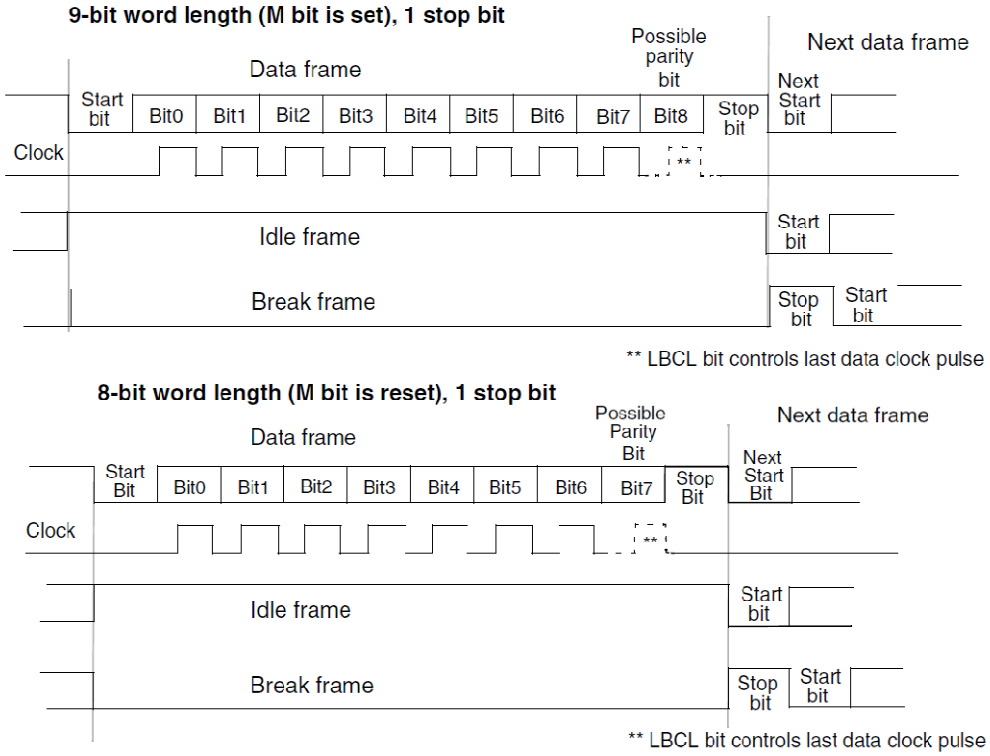
$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$$

PCLK 외 Baudrate 에 따르는 에러율 표이다. 에러율이 없는 값을 사용한다.
USART_InitStruct->USART_BaudRate = 115200; 전송속도 115200 이다.
USART1 는 PCLK2(36MHz)를 USART2,3,4,5 는 PCLK1(72MHz) 를 사용한다.

| Baud rate | | $f_{PCLK}$ = 36 MHz | | | $f_{PCLK}$ = 72 MHz | | |
|---|---|---|---|---|---|---|---|
| S.No | in Kbps | Actual | Value programmed in the Baud Rate register | % Error =(Calculated - Desired)B.Rate /Desired B.Rate | Actual | Value programmed in the Baud Rate register | % Error |
| 1. | 2.4 | 2.400 | 937.5 | 0% | 2.4 | 1875 | 0% |
| 2. | 9.6 | 9.600 | 234.375 | 0% | 9.6 | 468.75 | 0% |
| 3. | 19.2 | 19.2 | 117.1875 | 0% | 19.2 | 234.375 | 0% |
| 4. | 57.6 | 57.6 | 39.0625 | 0% | 57.6 | 78.125 | 0.% |
| 5. | 115.2 | 115.384 | 19.5 | 0.15% | 115.2 | 39.0625 | 0% |
| 6. | 230.4 | 230.769 | 9.75 | 0.16% | 230.769 | 19.5 | 0.16% |
| 7. | 460.8 | 461.538 | 4.875 | 0.16% | 461.538 | 9.75 | 0.16% |
| 8. | 921.6 | 923.076 | 2.4375 | 0.16% | 923.076 | 4.875 | 0.16% |
| 9. | 2250 | 2250 | 1 | 0% | 2250 | 2 | 0% |
| 10. | 4500 | NA | NA | NA | 4500 | 1 | 0% |

[그림 4.5.2 ] USART 의 Baudrate

2) Word Length 와 Parity




stm32f10x_usart.h 에 정의 되어 있다.

USART_InitStruct->USART_WordLength = USART_WordLength_8b;

⇨ WordLength_8b 은 7 비트 + Parity

USART_InitStruct->USART_WordLength = USART_WordLength_9b;

⇨ WordLength_9b 은 8 비트 + Parity

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

USART_InitStruct->USART_Parity = USART_Parity_Even ;
전송하는 data 에서 1 을 짝수로 만든다.
USART_InitStruct->USART_Parity = USART_Parity_Odd;
전송하는 data 에서 1 을 홀수로 만든다.
정상적인 data 를 전송 할려면 Parity_No 로 해야 한다.
USART_InitStruct->USART_Parity = USART_Parity_No ;

```
#define USART_Parity_No                      ((uint16_t)0x0000)
#define USART_Parity_Even                    ((uint16_t)0x0400)
#define USART_Parity_Odd                     ((uint16_t)0x0600)

#define USART_StopBits_1                     ((uint16_t)0x0000)
#define USART_StopBits_0_5                   ((uint16_t)0x1000)
#define USART_StopBits_2                     ((uint16_t)0x2000)
#define USART_StopBits_1_5                   ((uint16_t)0x3000)
```

3) USART_Mode

USART_InitStruct->USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

#define USART_Mode_Rx ((uint16_t)0x0004)
#define USART_Mode_Tx ((uint16_t)0x0008)

4) USART_Hardware_Flow_Control

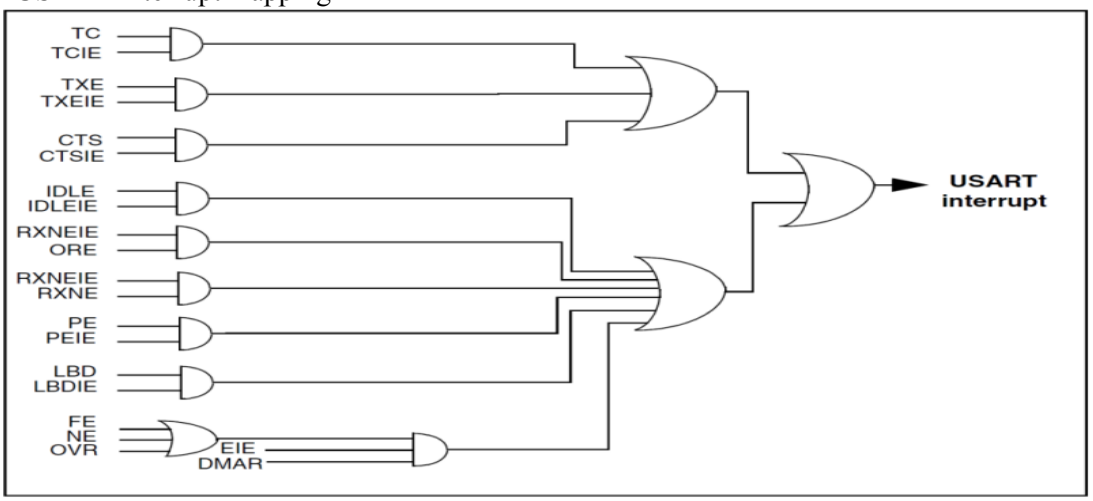
USART_InitStruct->USART_HardwareFlowControl = USART_HardwareFlowControl_None;
에서 RTS 나 CTS 를 하드웨어적으로 선을 연결하지 않기 때문에 USART_HardwareFlowControl_**None** 를 사용한다.

#define USART_HardwareFlowControl_None ((uint16_t)0x0000)
#define USART_HardwareFlowControl_RTS ((uint16_t)0x0100)
#define USART_HardwareFlowControl_CTS ((uint16_t)0x0200)
#define USART_HardwareFlowControl_RTS_CTS ((uint16_t)0x0300)

<USART Interrupt>
USART 는 총 9 개의 Interrupt Event 를 가지고 있다.
Event flag 는 SR 레지스터에 있으며, Enable control Bit 은 CR1 레지스터에 있다.

| Interrupt event | Event flag | Enable Control bit |
|---|---|---|
| Transmit Data Register Empty | TXE | TXEIE |
| CTS flag | CTS | CTSIE |
| Transmission Complete | TC | TCIE |
| Received Data Ready to be Read | RXNE | RXNEIE |
| Overrun Error Detected | ORE | |
| Idle Line Detected | IDLE | IDLEIE |
| Parity Error | PE | PEIE |
| Break Flag | LBD | LBDIE |
| Noise Flag, Overrun error and Framing Error in multibuffer communication | NE or ORE or FE | EIE |

USART Interrupt mapping




printf 를 사용할 때는 #include <stdio.h> 정의를 해 준다.

<STDIO 표준 입출력 함수>

| | |
|---|---|
| int getchar(void); | 문자를 입력 받는다. |
| char *gets(char *s); | 문자열을 입력 받는다. |
| int printf(const char *format, ...); | 형식화된 문자열을 출력한다. |
| int putchar(int c); | 문자를 출력한다. |
| int puts(const char *s); | 문자열을 출력한다. |
| int scanf(const char *format, ...); | 형식화된 문자를 입력 받는다. |
| int sprintf(char *s, const char *format, ...); | 형식화된 문자열을 만든다. |

#include <stdio.h> 정의를 해 주지 않고 직접 사용하는 경우도 있다.

```
void printf(const unsigned char *string)
{
        unsigned char temp;
        if(!DEBUG)
        {   return;         }
        temp = (unsigned char)(*string++);
        while(temp)
        {
                USART_SendData(USART1, temp);
                temp = (unsigned char)(*string++);
                while (!(USART1->SR & USART_FLAG_TXE));
        }
        while (!(USART1->SR & USART_FLAG_TC));
}

void putchar(unsigned char one_byte)
{
        if(!DEBUG)
        {   return;         }

        USART_SendData(USART1, one_byte);
        while (!(USART1->SR & USART_FLAG_TXE));
        while (!(USART1->SR & USART_FLAG_TC));
}

/*-------  SendChar        Write character to Serial Port.----------------*/
int SendChar (int ch)  {
  while (!(USART1->SR & USART_FLAG_TXE));
  USART1->DR = (ch & 0x1FF);
  return (ch);
}

/*---------  GetKey   Read character to Serial Port.-------------------*/
int GetKey (void)  {
  while (!(USART1->SR & USART_FLAG_RXNE));
  return ((int)(USART1->DR & 0x1FF));
}
```

### 4.6 ADC

Analog Digital Converter (ADC)는 Analog를 Digital로 변환하는 값으로 범위 0x0(0V)~0xFFF(3.3V) 까지를 12 비트 분해능을 갖춰다. Sampling Time 최대값은 17.1usec, Conversion시간은 최대 18usec 이다.
ADC 입력 clock은 PCLK2 의 72MHz 를 2,4,6,8 로 나누어 공급하고, 이 클럭의 최대값이 14 MHz 를 초과해서는 안된다. ADC 는 ADCx_IN0에서 ADCx_IN15의 16개의 포트를 사용 할 수 있다.

아래 그림을 보면 ADCx_IN0에서 ADCx_IN15 입력에서 GPIO Ports 다음 Analog MUX 에 Temp. Sensor 와 V REFINT 두 개의 핀이 더 있다.
Temp. Sensor 는 ADC_IN16 으로 STM32 내부에 온도센서를 자체에 있어 온도를 측정 할 수 있다.

Temperature (℃ ) = {(V25 - VSENSE) / Avg_Slope} + 25.
(V25 =1.43V, Avg_Slope =4.3mV/ ℃)
Temperature Range : –40 ~ 125도
Precision: ± 1.5 °C
V REFINT 는 ADC_IN17 로 Dual Mode 를 지원한다.

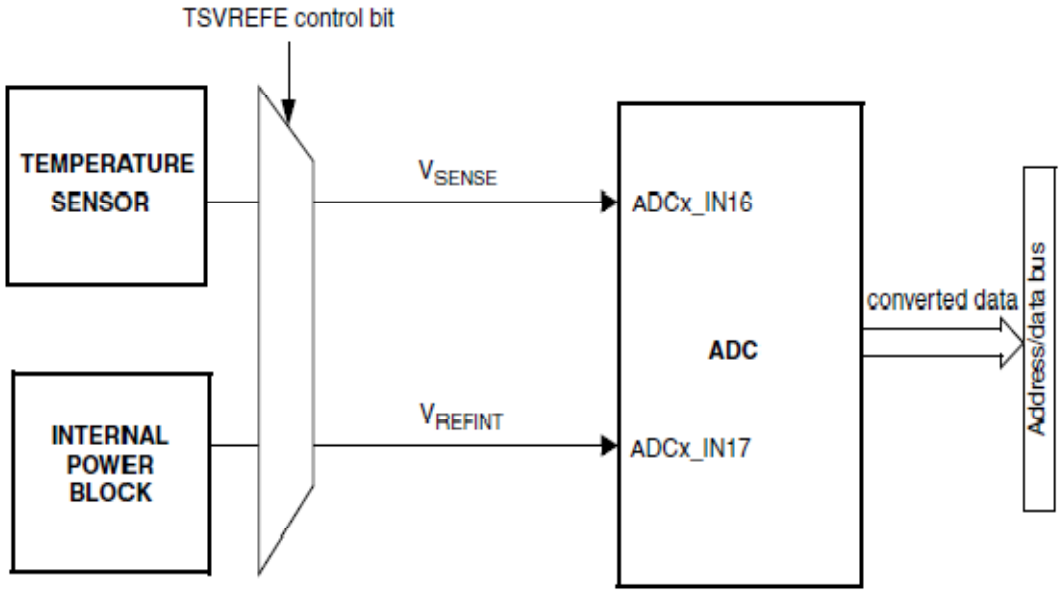



그림 Temperature sensor and VREFINT channel block diagram

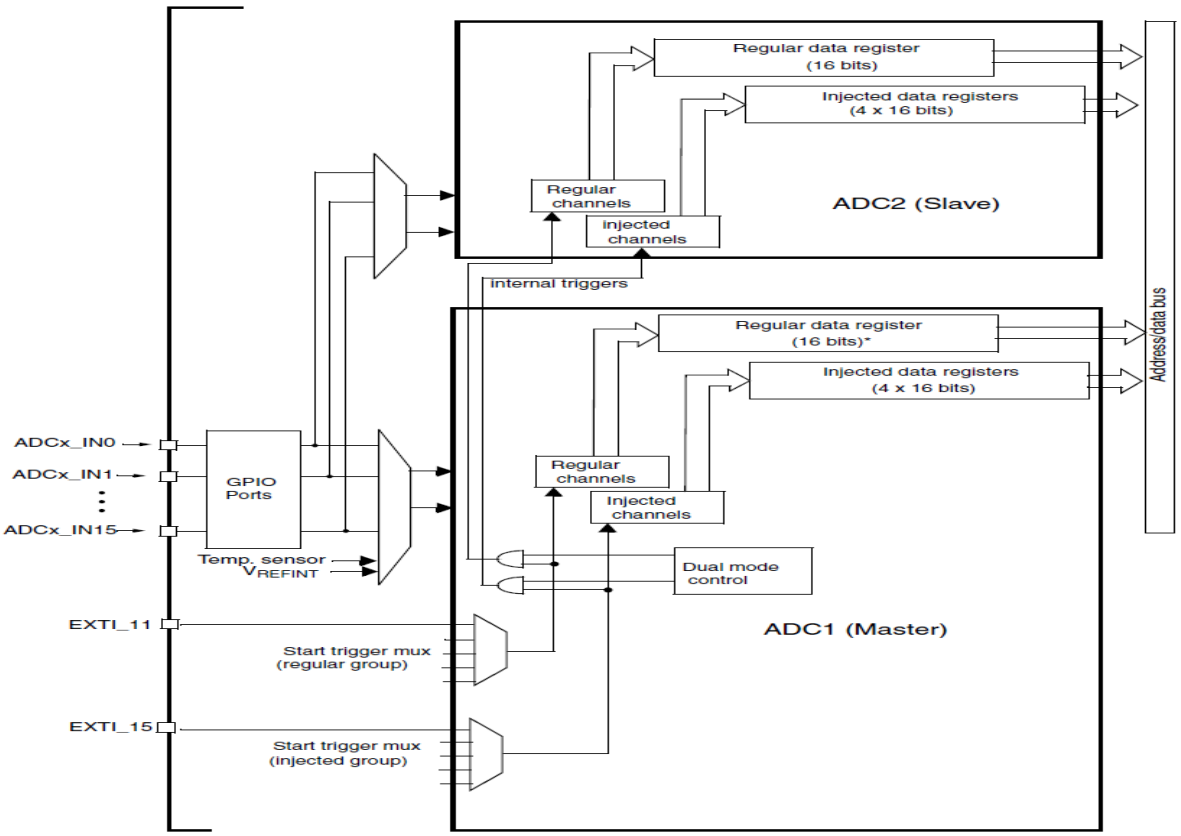
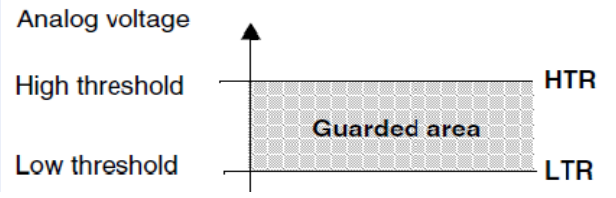



그림 Single ADC block diagram

위에서 Analog MUX 에 Temp. Sensor 와 V REFINT 두 개의 핀이 더 있는 것을 설명했다. Analog MUX 에서는 Regular channe 과, Injected channel 로 입력이 되는데 이곳에서 는 Analog를 Digital로 변환한다.
여러 개의 채널을 사용 할 때 먼저 처리해야 할 것을 정해야 하기 때문에 두 채널을 두어 Regular channel 보다 Injected channel 이 먼저 처리하게 되어 있다.
Injected channel는 Injected Data register로 Regular channel은, Regular Data register 로 연결되어 있다.

Analog watchdog
ADC 중에 Analog watchdog 가 있는데 Low Threshold 전압과 High Threshold 전압 범위를 벗어나면 인터럽트를 발생하는 것이다. 즉 아래 그림그림처럼 Guarded area 를 벗어나는 범위가 인터럽트 발생한다.



### 4.7 DAC

DAC 는 전압의 출력을 Digital 입력을 받아 analog 로 변환하는 기능으로 High Density Device이상에 주로 있고, 12비트 분해능을 지니고 있다.

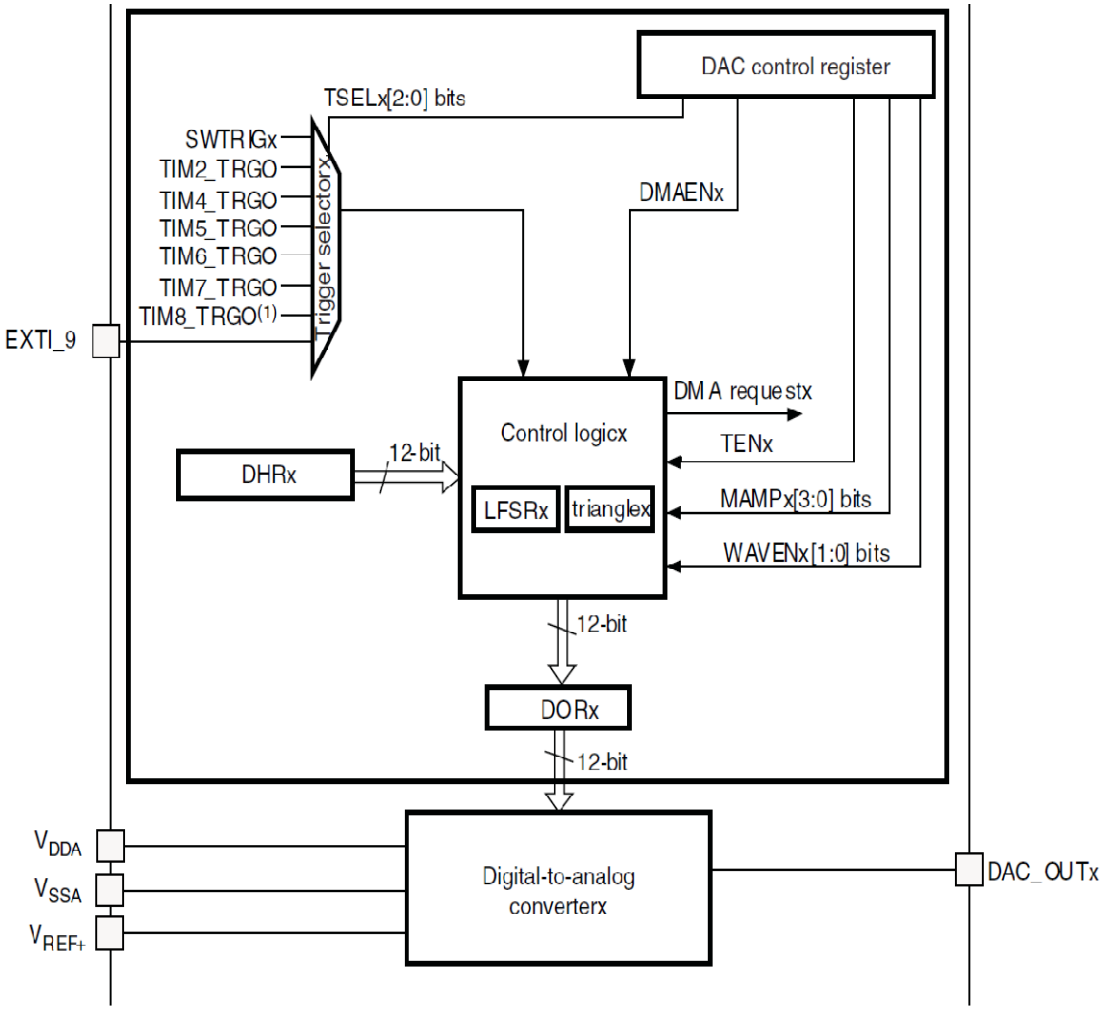



그림 DAC channel block diagram

GPIO pin (PA4 , PA5) 가 analog (AIN) 으로 사용된다..
PA4 는 SPI1_NSS/ USART2_CK/DAC_OUT1/ADC12_IN4
PA5 는 SPI1_SCK/ DAC_OUT2 /ADC12_IN5

PA4 , PA5는 ADC, DAC 둘 다 사용 가능한 핀이다.

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | DAC_CR | Reserved | | | DMAEN2 | MAMP2[3:0] | | | | WAVE2[2:0] | | TSEL2[2:0] | | | TEN2 | BOFF2 | EN2 | Reserved | | | DMAEN1 | MAMP1[3:0] | | | | WAVE1[2:0] | | TSEL1[2:0] | | | TEN1 | BOFF1 | EN1 |
| | Reset value | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x04 | DAC_SWTRIGR | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | SWTRIG2 | SWTRIG1 | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | |
| 0x08 | DAC_DHR12R1 | Reserved | | | | | | | | | | | | | | | | | | | | DACC1DHR[11:0] | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x0C | DAC_DHR12L1 | Reserved | | | | | | | | | | | | | | | | DACC1DHR[11:0] | | | | | | | | | | | | | Reserved | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| 0x10 | DAC_DHR8R1 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | DACC1DHR[7:0] | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x14 | DAC_DHR12R2 | Reserved | | | | | | | | | | | | | | | | | | | | DACC2DHR[11:0] | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x18 | DAC_DHR12L2 | Reserved | | | | | | | | | | | | | | | | DACC2DHR[11:0] | | | | | | | | | | | | | Reserved | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| 0x1C | DAC_DHR8R2 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | DACC2DHR[7:0] | | | | | | | |
| 0x20 | DAC_DHR12RD | Reserved | | | DACC2DHR[11:0] | | | | | | | | | | | | | Reserved | | | DACC1DHR[11:0] | | | | | | | | | | | | |
| | Reset value | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x24 | DAC_DHR12LD | DACC2DHR[11:0] | | | | | | | | | | | | Reserved | | | | DACC1DHR[11:0] | | | | | | | | | | | | | Reserved | | |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| 0x28 | DAC_DHR8RD | Reserved | | | | | | | | | | | | | | | | DACC2DHR[7:0] | | | | | | | | DACC1DHR[7:0] | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x2C | DAC_DOR1 | Reserved | | | | | | | | | | | | | | | | | | | | DACC1DOR[11:0] | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x30 | DAC_DOR2 | Reserved | | | | | | | | | | | | | | | | | | | | DACC2DOR[11:0] | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

그림 DAC register map

아래 프로그램은 PA4, PA5 포트에서 크기가 다른 삼각펄스를 만든다.

```
/* TIM2 Configuration */
   TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
   TIM_TimeBaseStructure.TIM_Period = 0xF;
```

```
  TIM_TimeBaseStructure.TIM_Prescaler = 0xF;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

  /* TIM2 TRGO selection */
  TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);

  /* DAC channel1 Configuration */
  DAC_InitStructure.DAC_Trigger = DAC_Trigger_T2_TRGO;
  DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_Triangle;
  DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude                    =
  DAC_TriangleAmplitude_2047;
  DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Disable;
  DAC_Init(DAC_Channel_1, &DAC_InitStructure);

  /* DAC channel2 Configuration */
  DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude                    =
  DAC_TriangleAmplitude_1023;
  DAC_Init(DAC_Channel_2, &DAC_InitStructure);

  /* Enable DAC Channel1: Once the DAC channel1 is enabled, PA.04 is
      automatically connected to the DAC converter. */
  DAC_Cmd(DAC_Channel_1, ENABLE);

  /* Enable DAC Channel2: Once the DAC channel2 is enabled, PA.05 is
      automatically connected to the DAC converter. */
  DAC_Cmd(DAC_Channel_2, ENABLE);

  /* Set DAC dual channel DHR12RD register */
  DAC_SetDualChannelData(DAC_Align_12b_R, 0x100, 0x100);

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

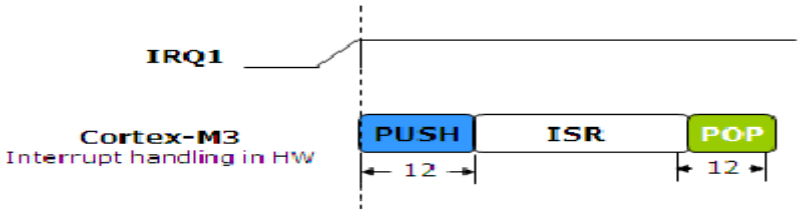
위 프로그램 소스는 14_DAC 에 있다.

## 4.8 NVIC

STM32 의 Interrupt 는 Cortex-M3 Core 의 NVIC 를 통해서 제어한다.

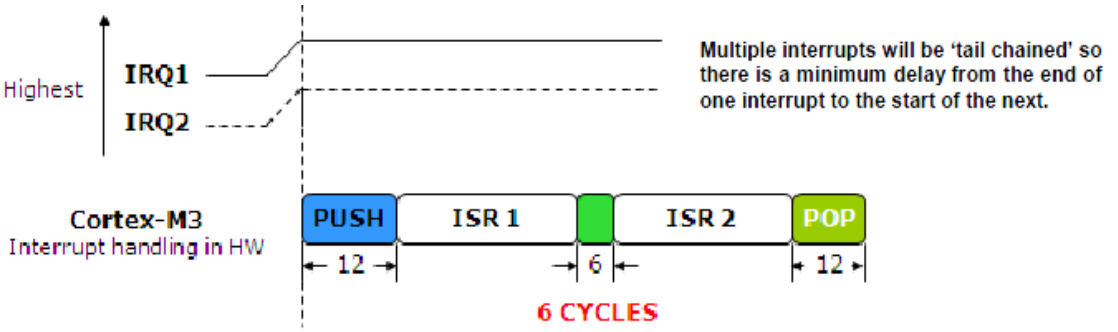
### 4.8.1 Low-latency interrupt handling

CORTEX-M3 에서는 인터럽트 발생 시 Stack 에 사용 중이던 레지스터를 push 하고, Vector table 의 해당 인터럽트 벡터로 jump 하는 동작을 HW 적으로 처리한다. 인터럽트 수행이 끝나고 Pop 하는 동작도 HW 적으로 처리한다. HW 적으로 Stacking 동작이 처리되니 빠르기도 하겠지만, Tail chaining 이나 Late Arrival 같은 기술도 함께 적용되어 있다. 인터럽트 발생시 HW 적으로 12Cycle 만에 Stack push, pop 동작을 수행한다.

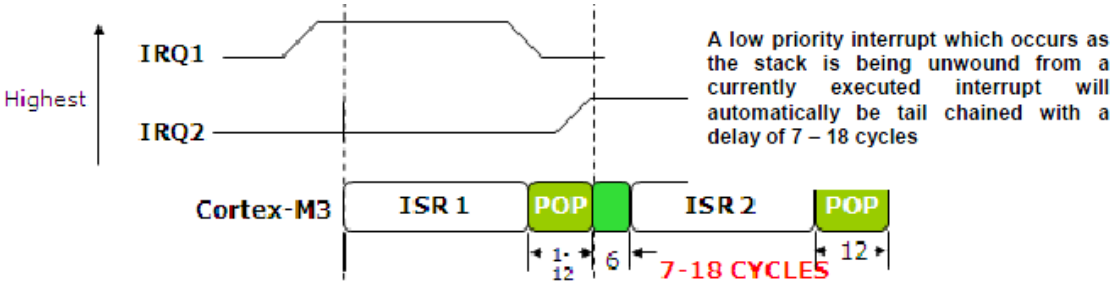



### 4.8. 2 Tail Chaining

우선순위가 높은 IRQ1 이 수행 중 IRQ2 의 인터럽트가 발생하였을 경우,
IRQ1 의 인터럽트가 완료되면, 바로 IRQ2 의 인터럽트가 수행이다.
이때 IRQ1 에서 IRQ2 로 전환될 때 불필요한 Stack 처리 동작없이 6Cycles 만에 전환되도록 하였다.

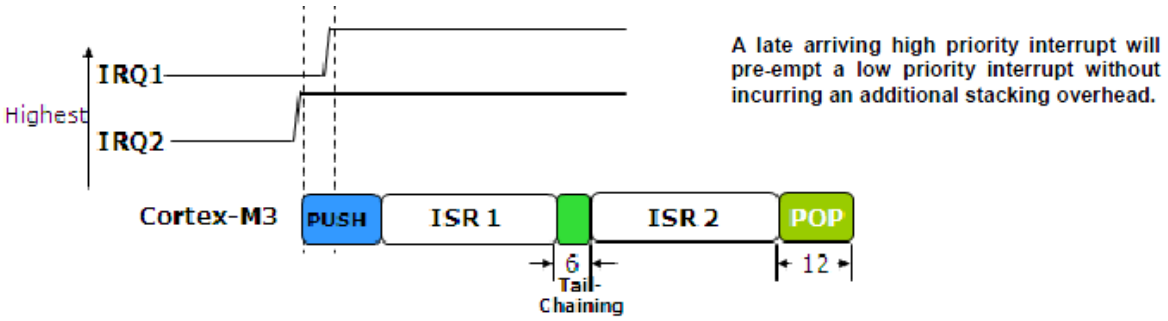



만약, 아래 그림처럼 IRQ1 이 종료되고 POP 동작 수행 중에 IRQ2 가 발생될 경우에는, POP 동작을 취소하고 바로 IRQ2 로 진입한다.
이 경우에도 IRQ2 발생 시점을 기준으로 놓고 보면 Latency 는 6Cycle 이라고 할 수 있다.



### 4.8.3 Late Arrival

IRQ2 를 위해서 PUSH 동작 중 우선순위가 높은 IRQ1 이 발생하면 PUSH 동작 완료 후 바로 IRQ1 이 수행이다. IRQ1 이 완료되면 그때 IRQ2 가 수행한다.




### 4.8.4 4Bit Priority

STM32 는 Interrupt 의 우선순위를 위해서 4bit Priority 설정을 사용하며,
Group Priority 와 Subpriority 라는 값을 사용한다.
4Bit 의 Priority bit 은 PRIGROUP 의 설정에 따라서 Group priority 와 Subpriority 에서 사용하는 비트의 비율이 달라진다.
아래 표에서 PRIGROUP 이 0b100 인 경우를 보면, 이 경우 4 비트 중 상위 3 비트(7:5 번비트)는 Group priority 에서 사용하며 하위 1 비트(4 번비트)는 Subpriority 에서 사용한다.

**Priority grouping**

| PRIGROUP [2:0] | Interrupt priority level value, PRI_*N*[7:4] | | | Number of | |
|---|---|---|---|---|---|
| | Binary point(1) | Group priority bits | Subpriority bits | Group priorities | Sub priorities |
| 0b011 | 0bxxxx | [7:4] | None | 16 | None |
| 0b100 | 0bxxx.y | [7:5] | [4] | 8 | 2 |
| 0b101 | 0bxx.yy | [7:6] | [5:4] | 4 | 4 |
| 0b110 | 0bx.yyy | [7] | [6:4] | 2 | 8 |
| 0b111 | 0b.yyyy | None | [7:4] | None | 16 |

Group Priority 는 Pre-emption 의 기준이 되는 우선 순위이다.
만약 Group Priority 의 우선순위가 낮은 ISR1 이 실행 중 더 높은 Group priority 값을 가지는 ISR2 이 발생하면, ISR2 가 CPU 를 선점한다.
Subpriorit 는 같은 Group Priority 의 인터럽트가 pending 된 시점에서 어떤 인터럽트를 먼저 실행하는 기준이 되는 값이다.

### 4.8.5 Vector Table

STM32 는 Vector Table 에 코드가 위치하는 것이 아닌 함수의 포인터 값이 위치한다. STM32 의 각 Peripheral 들의 Vector Table Index 는 다음 표와 같이 미리 정의되어 있다. 음영 처리된 부분이 Cortex-M3 의 기본 Exception 이며,
이 부분을 제외한 나머지는 STM32 의 제품 군 마다 peripheral 의 구성이 다르기 때문에,Vector Table 의 구조도 달라질 수 있다.

| Position | Priority | Type of priority | Acronym | Description | Address |
|---|---|---|---|---|---|
| | - | - | - | Reserved | 0x0000_0000 |
| | -3 | fixed | Reset | Reset | 0x0000_0004 |
| | -2 | fixed | NMI | Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector. | 0x0000_0008 |
| | -1 | fixed | HardFault | All class of fault | 0x0000_000C |
| | 0 | settable | MemManage | Memory management | 0x0000_0010 |
| | 1 | settable | BusFault | Prefetch fault, memory access fault | 0x0000_0014 |
| | 2 | settable | UsageFault | Undefined instruction or illegal state | 0x0000_0018 |
| | - | - | - | Reserved | 0x0000_001C - 0x0000_002B |
| | 3 | settable | SVCall | System service call via SWI instruction | 0x0000_002C |
| | 4 | settable | Debug Monitor | Debug Monitor | 0x0000_0030 |
| | - | - | - | Reserved | 0x0000_0034 |
| | 5 | settable | PendSV | Pendable request for system service | 0x0000_0038 |
| | 6 | settable | SysTick | System tick timer | 0x0000_003C |
| 0 | 7 | settable | WWDG | Window watchdog interrupt | 0x0000_0040 |
| 1 | 8 | settable | PVD | PVD through EXTI Line detection interrupt | 0x0000_0044 |
| 2 | 9 | settable | TAMPER | Tamper interrupt | 0x0000_0048 |
| 3 | 10 | settable | RTC | RTC global interrupt | 0x0000_004C |
| 4 | 11 | settable | FLASH | Flash global interrupt | 0x0000_0050 |
| 5 | 12 | settable | RCC | RCC global interrupt | 0x0000_0054 |
| 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 |
| 11 | 18 | settable | DMA1_Channel1 | DMA1 Channel1 global interrupt | 0x0000_006C |
| 12 | 19 | settable | DMA1_Channel2 | DMA1 Channel2 global interrupt | 0x0000_0070 |
| 13 | 20 | settable | DMA1_Channel3 | DMA1 Channel3 global interrupt | 0x0000_0074 |
| 14 | 21 | settable | DMA1_Channel4 | DMA1 Channel4 global interrupt | 0x0000_0078 |
| 15 | 22 | settable | DMA1_Channel5 | DMA1 Channel5 global interrupt | 0x0000_007C |
| 16 | 23 | settable | DMA1_Channel6 | DMA1 Channel6 global interrupt | 0x0000_0080 |

### 4.8.6 NVIC Register

NVIC Register 의 Base Address 는 0xE000E100 이다.

| Register | Name | Desc. |
|---|---|---|
| ISER | Interrupt Set Enable Register | 인터럽트를 set 상태로 설정 |
| ICER | Interrupt Clear Enable Register | 인터럽트를 Clear 상태로 설정 |
| ISPR | Interrupt Set Pending Register | 강제로 인터럽트를 Pending 상태로 설정할 수 있으며, 현재 Pending 상태를 읽을 수 있다. |
| ICPR | Interrupt Clear Pending Register | 강제로 인터럽트의 Pending 상태를 Clear 할 수 있으며, 현재 Pending 상태를 읽을 수 있다. |
| IABR | Interrupt Active Bit Register | 실제 인터럽트가 Active 된 상태인지 읽을 수 있다. |
| IP | Interrupt Priority Register | 개별 인터럽트 채널의 Priority 를 설정한다. |
| STIR | Software Trigger Interrupt Register | STIR 에 인터럽트 ID 를 Write 하면 SW 적으로 인터럽트를 발생한다. |

### 4.8.7 System Control Block(SCB) Register

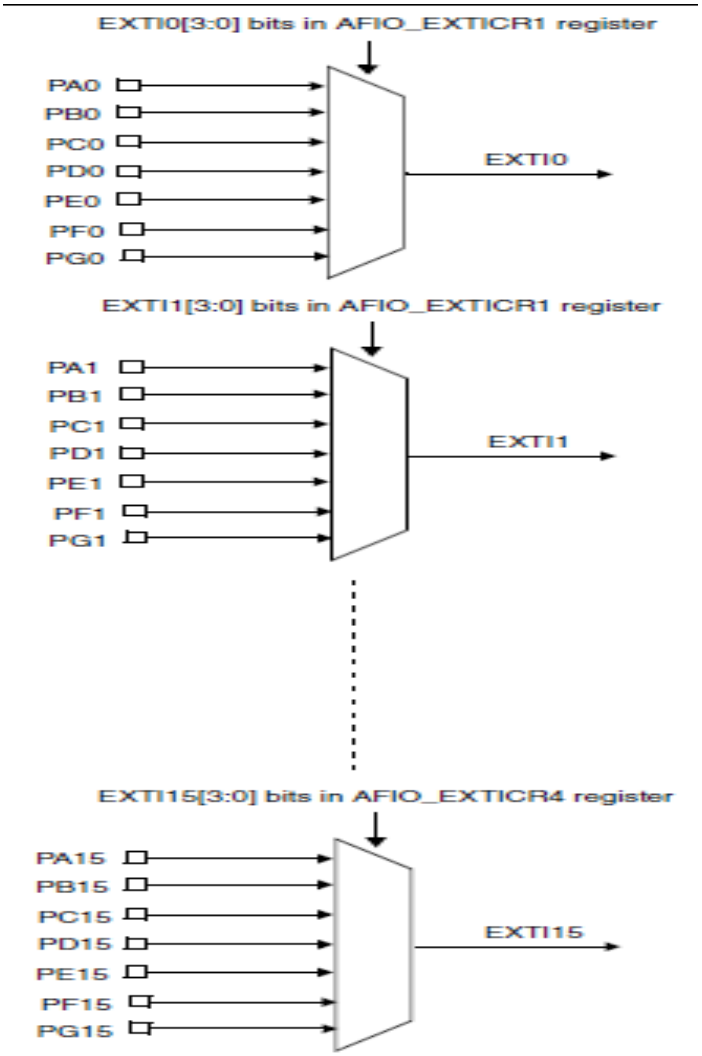
SCB 는 시스템 관련 레지스터들의 블록이다. SCB 의 Base Address 는 0xE000ED00 이다.

| Register | Name | Desc. |
|---|---|---|
| VTOR | Vector Table Offset Register | Vector Table 의 위치를 변경할 수 있다. |
| AIRCR | Application Interrupt / Reset Control Register | PRIGROUP 값을 설정하여, Priority Grouping 정책을 결정한다. |

### 4.8.8 External Interrupt

External Interrupt 는 GPIO 핀을 통해서 발생하는 인터럽트를 의미한다.

<External Interrupt GPIO Mapping>
STM32 는 EXTI0 부터 EXTI15 까지 총 16 개의 외부 인터럽트를 지원하며,
특정 GPIO 핀 16 개로 고정되지 않고 모든 핀이 인터럽트 소스로 사용될 수 있다.
그러나 인터럽트 소스로 사용될 수 있는 GPIO 그룹은 정해져 있다.
다음 그림과 같이 각 포트의 0 핀은 EXTI0 인터럽트로 사용되고, 각 포트의 1 번 핀은 EXTI1 인터럽트로 사용이다. 즉, PA0 와 PB0 는 동시에 인터럽트 소스로 사용할 수 없다.




EXTI 인터럽트 소스로 어떤 핀을 사용할지는, AFIO 레지스터의 EXTICR 레지스터에서 설정한다. 아래 EXTICR1 레지스터를 보면 EXTI0 부터 EXTI3 에 대한 GPIO source pin 을 설정하게 되어있다.
이런 식으로 EXTICR1~EXTICR4 까지 총 4 개의 레지스터에서 16 개의 핀을 설정한다.

### External interrupt configuration register 1 (AFIO_EXTICR1)

Address offset: 0x08

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 |

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]:** EXTI x configuration (x= 0 to 3)

These bits are written by software to select the source input for EXTIx external interrupt.
Refer to *Section 9.2.5: External interrupt/event line mapping on page 191*

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

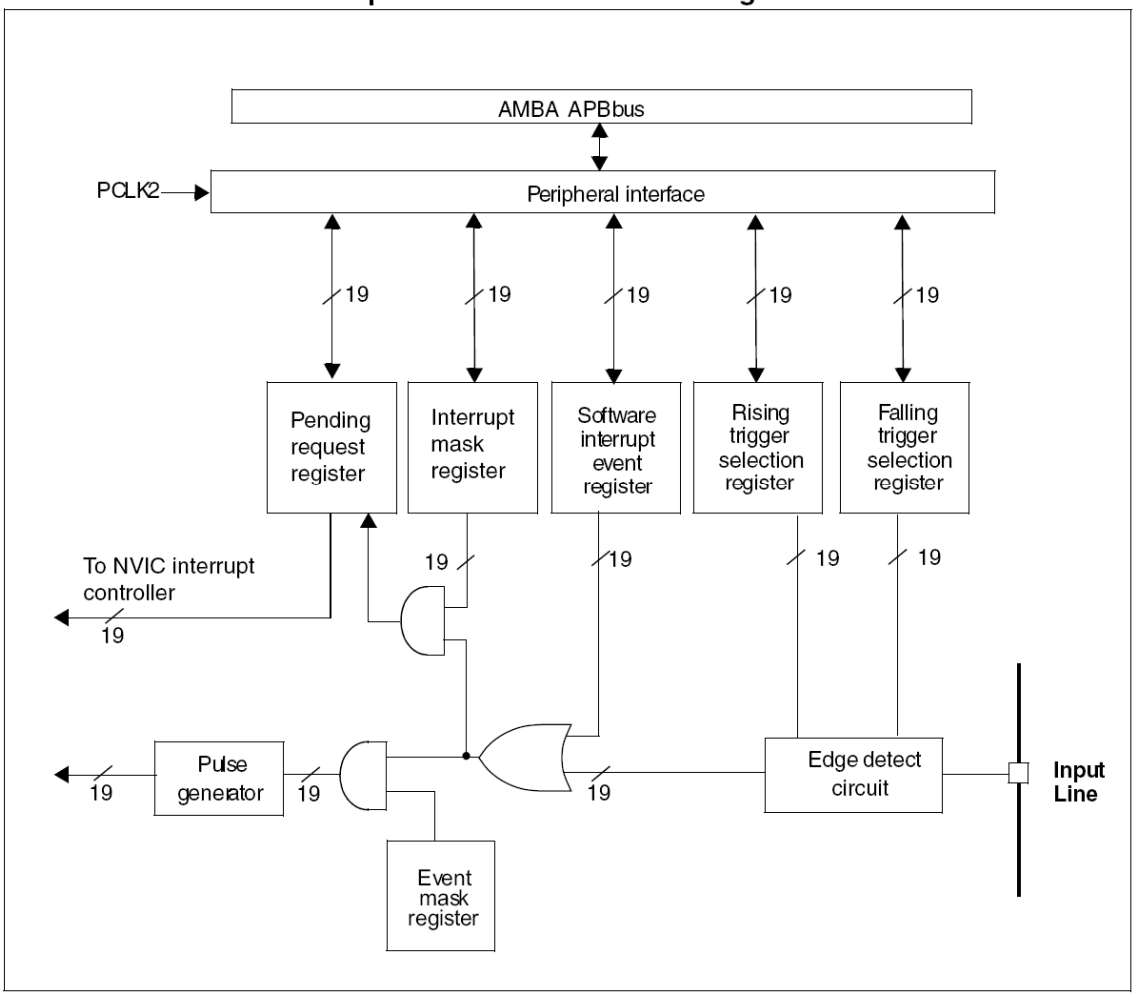
참고로 EXTI 인터럽트 소스는 EXTI19 까지 있다.
EXTI16~EXTI19 은 다음과 같이 사용이다.

| | |
|---|---|
| EXTI0~EXTI16 | GPIO 0~16 핀 인터럽트 소스 |
| EXTI16 | PVD output |
| EXTI17 | RTC alarm event |
| EXTI18 | USB Wakeup event |
| EXTI19 | Ethernet Wakeup event |

### 4.8.9 External interrupt Control Block

아래 그림만 이해하면 STM32 의 External interrupt 가 어떻게 동작하는지 이해할 수 있다.

**External interrupt/event controller block diagram**




1. Input Line 에 신호가 들어오면 Edge detect circuit 에서 Falling 또는 Rising 를 검출한다.

2. 위의 1 번에서 들어온 신호와 Software interrupt event register 를 통해서 인터럽트를 발생 시킨다.

3. 2 번에서 나온 신호가 Pending Request Register 에 Set 이다. 동시에 Event mask Register 설정에 따라서 이벤트 신호로도 사용될 수 있다. 이벤트는 Wakeup 신호이며, Event mask register 를 설정하면 인터럽트 신호를 wakeup 신호로 설정할 수 있다.

4. 인터럽트가 Pending 되고, interrupt mask register 가 설정되면 NVIC 에 의해서 인터럽트가 발생이다.

### 4.8.10 AFIO Register

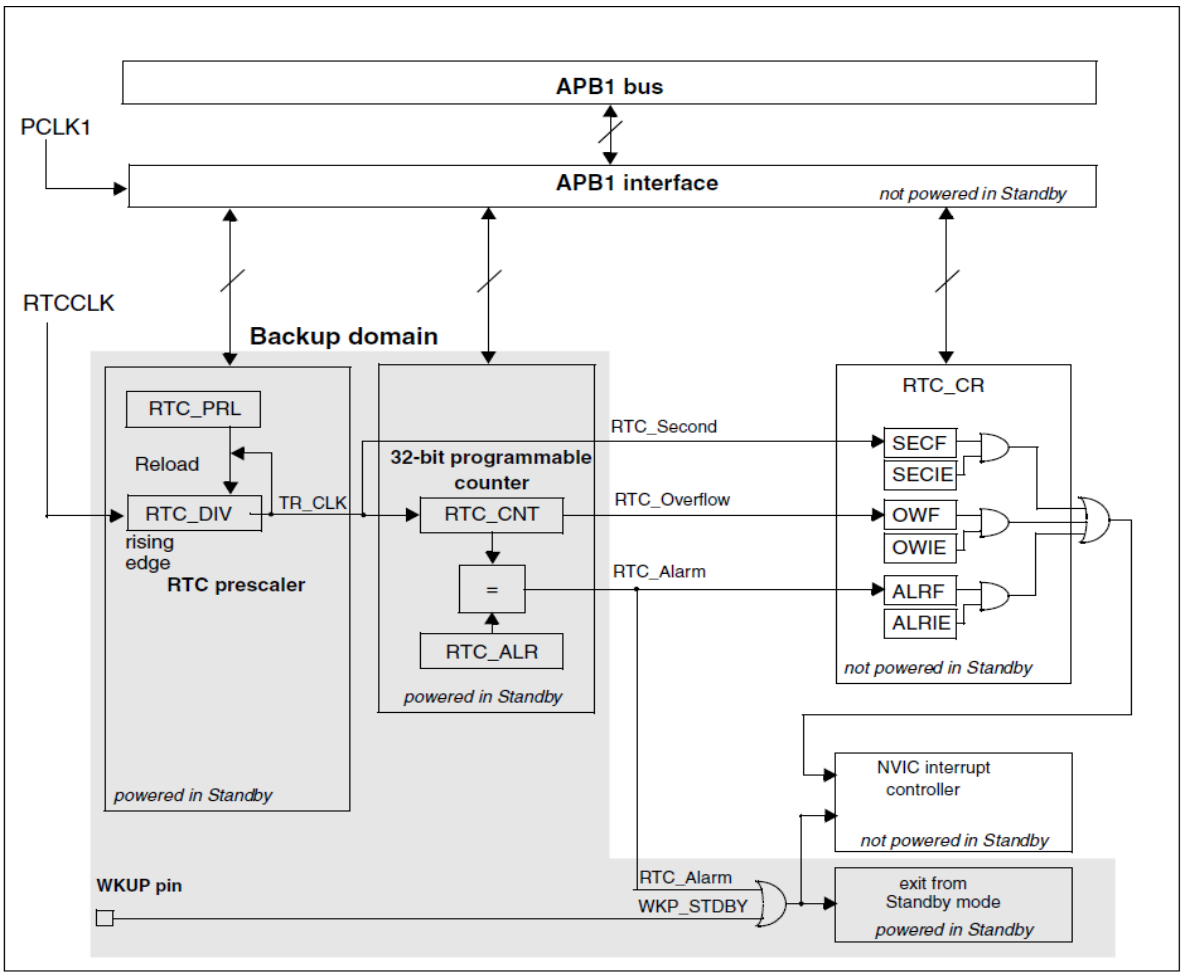
| | | |
|---|---|---|
| EXTICR1 | External interrupt configuration Register | EXTI0~EXTI3 의 인터럽트 소스 설정 |
| EXTICR2 | External interrupt configuration Register | EXTI4~EXTI7 의 인터럽트 소스 설정 |
| EXTICR3 | External interrupt configuration Register | EXTI8~EXTI11 의 인터럽트 소스 설정 |
| EXTICR4 | External interrupt configuration Register | EXTI12~EXTI15 의 인터럽트 소스 설정 |

### 4.8.11 EXTI Register

| | | |
|---|---|---|
| EXTI_IMR | Interrupt Mask Register | 인터럽트 Enable/Disable 설정 |
| EXTI_EMR | Event Mask Register | 이벤트 Enable/Disable 설정 |
| EXTI_RTSR | Rising Trigger Selection Register | Rising Edge 설정 |
| EXTI_FTSR | Falling Trigger Selection Register | Falling Edge 설정 |
| EXTI_SWIER | Software Interrupt Event Register | SW 적으로 인터럽트, 이벤트 발생 |
| EXTI_PR | Pending Register | Pending 상태 정보 레지스터 |

## 4.9 RTC

RTC (Real Time Clock) 인터럽트 지원 (아래 그림 중간 참고 )
1) Alarm 인터럽트 설정
2) 주기적인 Second 인터럽트 발생
3) 오버플로 인터럽트 발생




RTC - register map and reset values

STM32 는 성능라인과 액세스라인이라는 서로 다른 두 개의 측면을 가지고 있다. 성능 라인은 최대 72MHz로 작동하는 주변장치와 액세스 라인은 최대 32MHz로 작동하는 주변장치를 가지고 있다.

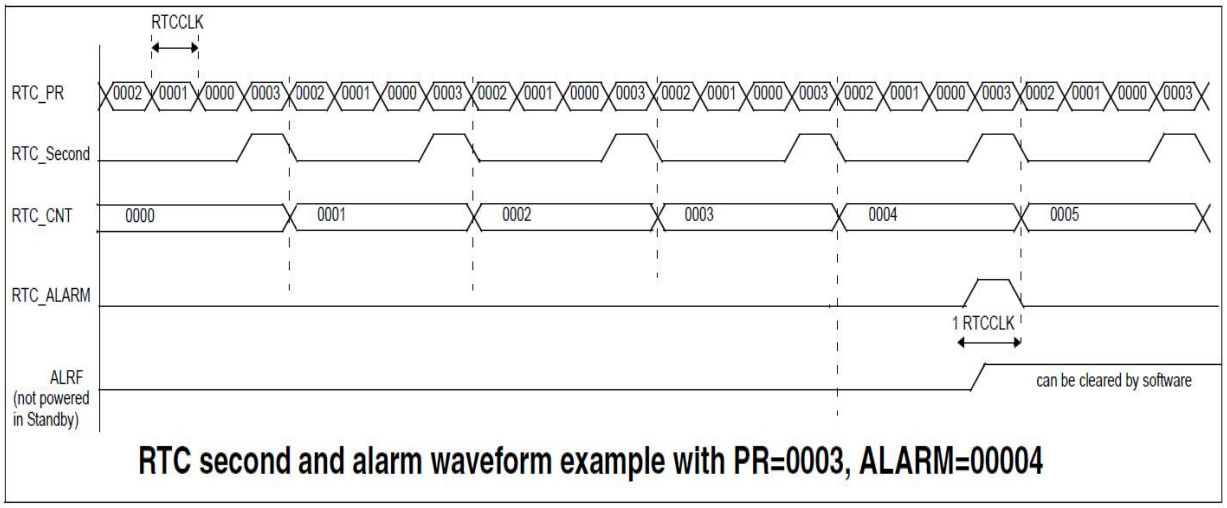
APB1 Interface 는 PCLK1 를 받아 APB1 bus 를 통한 16 비트 레지스터를 Read, Write 한다. RTCCLK 를 받아 20 Bit 의 RTC_DIV 를 지나 TR_CLK 을 만들어 RTC_Second (1sec) 를 만드는 부분을 RTC prescaler 이다.

```
/* Set RTC prescaler: set RTC period to 1sec */
RTC_SetPrescaler(32767);
/* RTC period = RTCCLK/RTC_PR = (32.768 KHz)/(32767+1) */
```

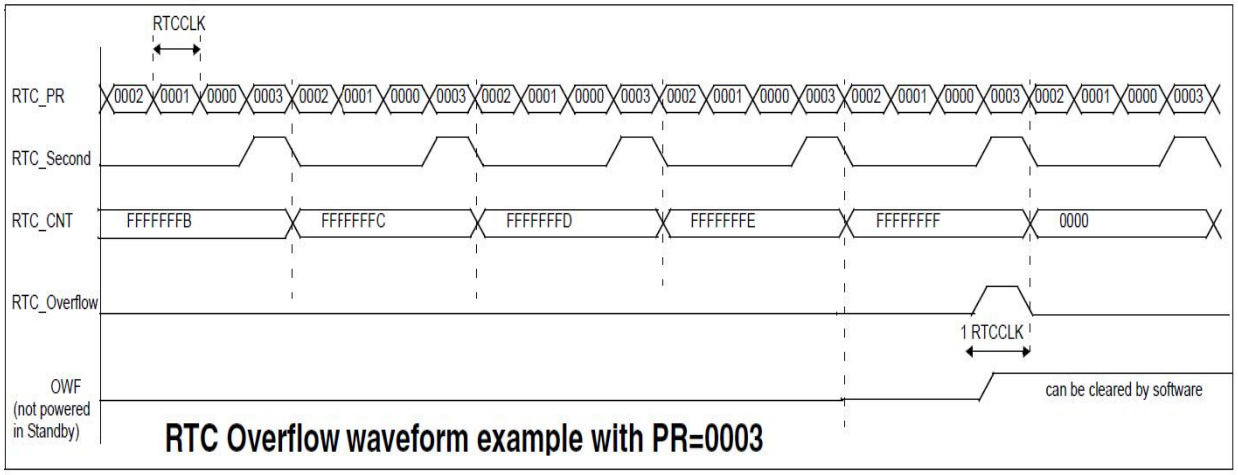
다음으로 32-Bit programmable counter 부분은 RTC_PRL, RTC_ALR 로 구성되어 초기화와 관련이 있다.

RTC_PRL, RTC_ALR, RTC_CNT, RTC_DIV 레지스터들은 System Reset 이나 Power Reset 에 의해서 reset 되지 않고, Backup Domain reset 에 의해서만 리셋이 된다.

RTC simplified block diagram



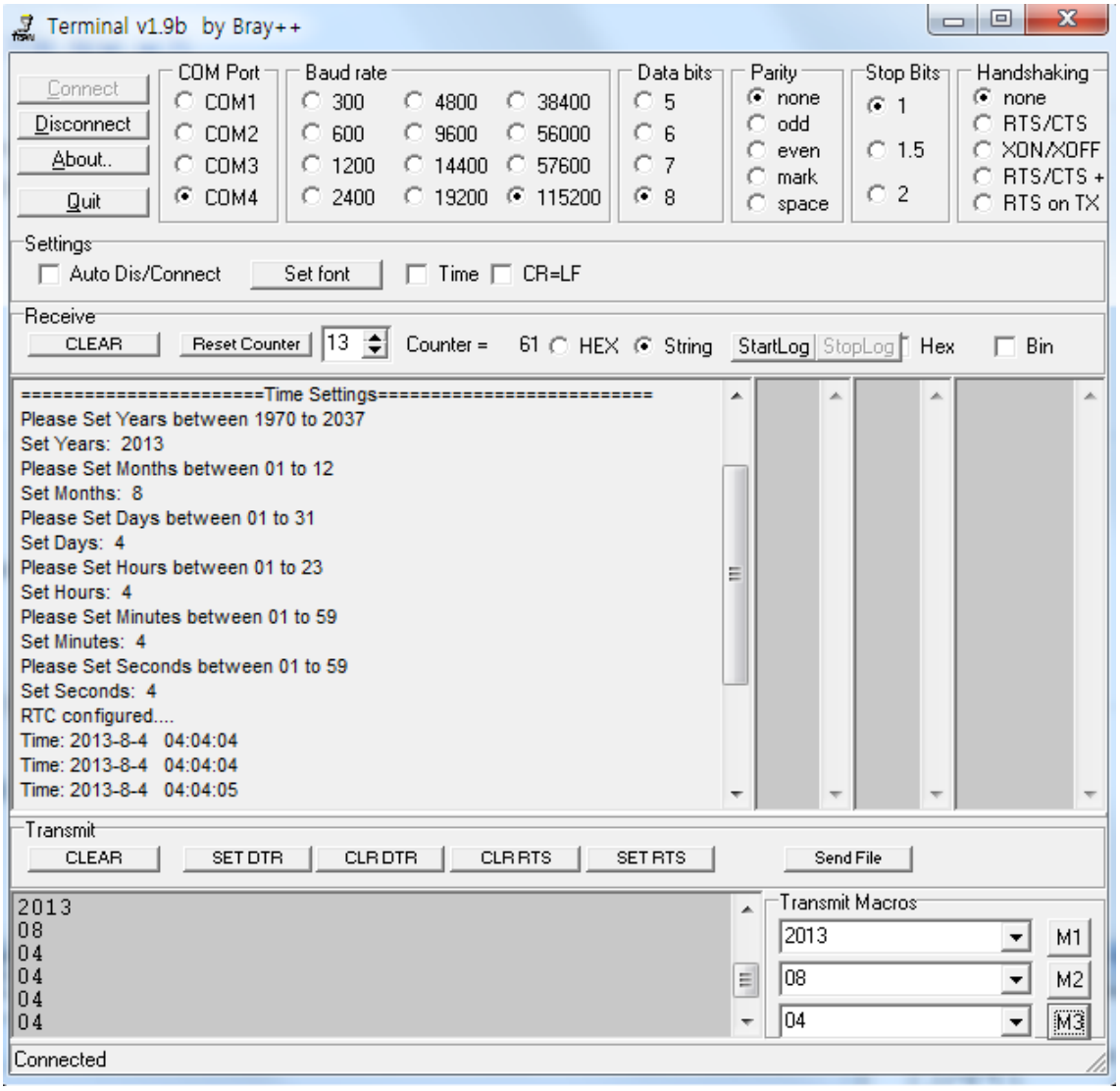

RTC 카운터값이 4에 도달하는 순간에(갱신전) Second 인터럽트 나 Alarm 인터럽트 발생한다.



| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x000 | RTC_CRH | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | OWIE | ALRIE | SECIE |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 |
| 0x004 | RTC_CRL | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | RTOFF | CNF | RSF | OWF | ALRF | SECF |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | 0 | 0 | 0 | 0 | 0 |
| 0x008 | RTC_PRLH | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | PRL[19:16] | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 |
| 0x00C | RTC_PRLL | Reserved | | | | | | | | | | | | | | | | PRL[15:0] | | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x010 | RTC_DIVH | Reserved | | | | | | | | | | | | | | | | DIV[31:16] | | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x014 | RTC_DIVL | Reserved | | | | | | | | | | | | | | | | DIV[15:0] | | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x018 | RTC_CNTH | Reserved | | | | | | | | | | | | | | | | CNT[13:16] | | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x01C | RTC_CNTL | Reserved | | | | | | | | | | | | | | | | CNT[15:0] | | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x020 | RTC_ALRH | Reserved | | | | | | | | | | | | | | | | ALR[31:16] | | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x024 | RTC_ALRL | Reserved | | | | | | | | | | | | | | | | ALR[15:0] | | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

다음은 RTC 프로그램(프로그램=> 11_RTC)을 실행했을 경우이다.

보드 (J14)에 PC 포트에 연결한다. 아래 그림은 통신포트 COM4, 속도 115200에 통신설정되어있다. (COM4는 연결할 때 확인한다. COM4가 아닐 수도 있다.)




실행결과

```
**********************************************************
*     RTC Test program ! ^_^    *
*                                                        *
RTC not yet configured....
==========Time Settings==========
Please Set Years between 1970 to 2037
Set Years:    2013
Please Set Months between 01 to 12
Set Months:    8                    <= 08 입력   ** 한 자릿수는 0을 붙인다.
다른것도 동일
Please Set Days between 01 to 31
Set Days:    4                      <= 04 입력
Please Set Hours between 01 to 23
Set Hours:    4
```

Please Set Minutes between 01 to 59
Set Minutes:  4
Please Set Seconds between 01 to 59
Set Seconds:  4
RTC configured....
Time: 2013-8-4  04:04:04
Time: 2013-8-4  04:04:05
Time: 2013-8-4  04:04:06
…..

## 4.10 TIM

### 4.10.1. TIM_IRQ Channels

STM 32F103Rx 에는 1 개의 Advanced-Control 타이머(TIM1)로 주로 PWM 펄스 생성하며, 3 개의 General-Purpose 타이머(TIM2,3,4)를 내장하고 있다. 4 개의 타이머는 모두 독립적이며, 타이머 사이의 동기화를 위해서 master/slave mode 라는 기능을 제공한다.

다음 표를 보시면, TIM1 과 TIM2,3,4 의 차이는 Complementary Output 의 지원 차이다. Complementary outputs 기능은 PWM 의 기능을 좀더 보완해주는 기능 이다.

**Timer feature comparison**

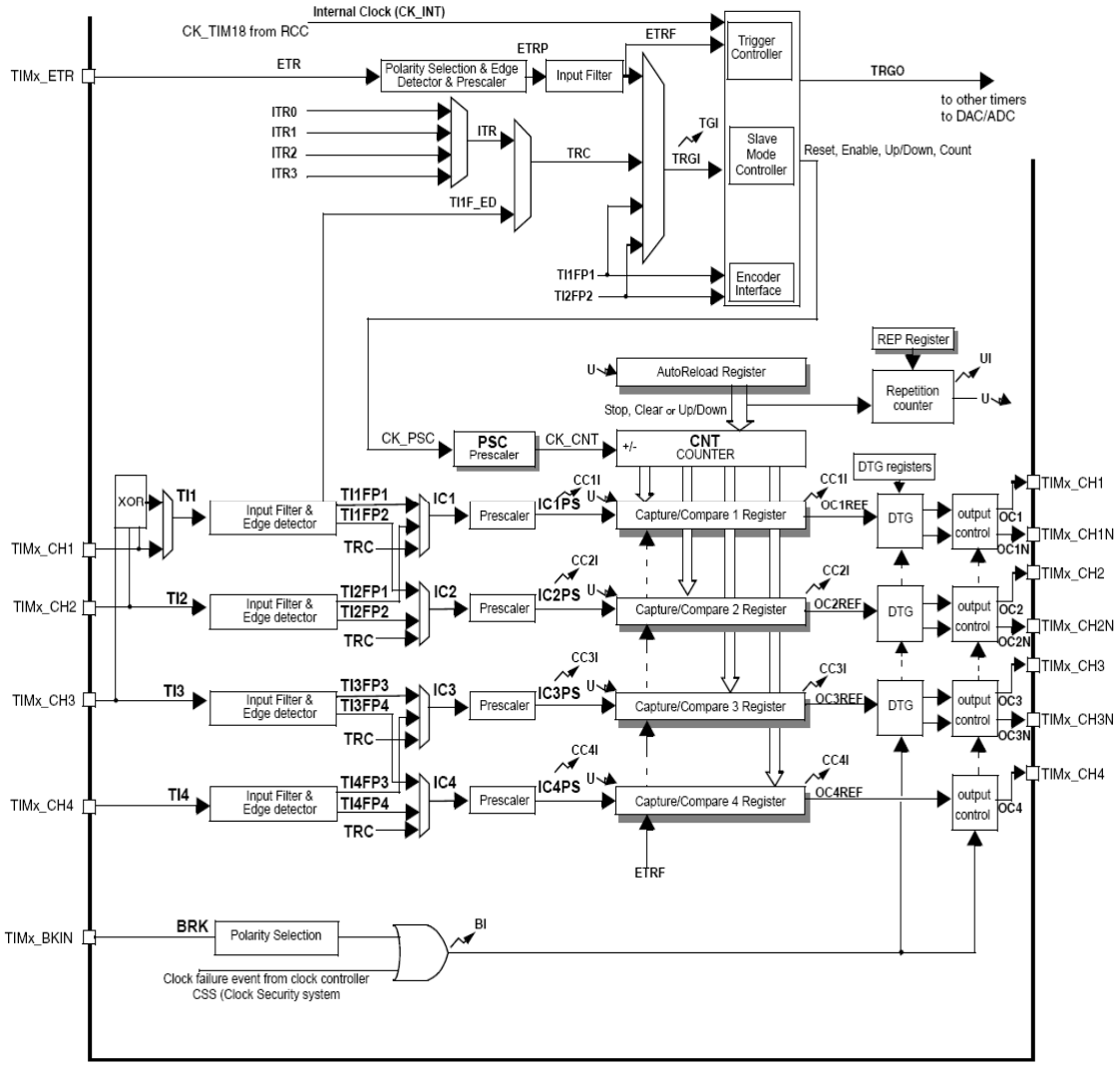
| Timer | Counter resolution | Counter type | Prescaler factor | DMA request generation | Capture/compare channels | Complementary outputs |
|---|---|---|---|---|---|---|
| TIM1 | 16-bit | Up, down, up/down | Any integer between 1 and 65536 | Yes | 4 | Yes |
| TIM2, TIM3, TIM4 | 16-bit | Up, down, up/down | Any integer between 1 and 65536 | Yes | 4 | No |

Gerneral Purpose 타이머의 주요 특징은 다음과 같다.

• 16-bit up, down, up/down auto-reload counter.
• 16-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency either by any factor between 1 and 65535.
• Up to 4 independent channels for:
– Input Capture
– Output Compare
– PWM generation (Edge and Center-aligned Mode)
– One-pulse mode output
• Complementary outputs with programmable dead-time
• Synchronization circuit to control the timer with external signals and to interconnect several timers together.
• Repetition counter to update the timer registers only after a given number of cycles of the counter.
• Break input to put the timer’s output signals in reset state or in a known state.
• Interrupt/DMA generation on the following events:
– Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
– Trigger event (counter start, stop, initialization or count by internal/external trigger)
– Input capture
– Output compare
– Break input
• Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
• Trigger input for external clock or cycle-by-cycle current management

타이머의 전체적인 동작에 대한 블록도이다.

**Advanced-control timer block diagram**

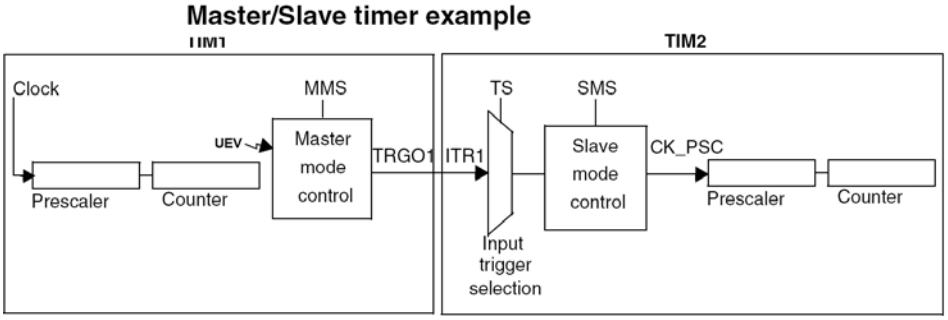



위 블록도는 다음과 같다.

1. 그림 상단 중앙에 타이머 Clock(CK_INT)은 Prescaler(PSC)에 의해서 분주되어 카운터 레지스터를 증가시키며, 이 카운터 레지스터의 값이 Auto-reload 레지스터의 값과 같아지면 Update 이벤트가 발생하는 것이다.
여기서 Clock(CK_INT) 클럭 공식은 fCK_PSC/ (PSC[15:0] +1) 이다.

2. Trigger 이벤트는 타이머의 동기화를 지원하는 기능이다. Trigger 의 외부 입력 핀 신호로는 ETR, TI1F_ED, TI1FP1, TI2FP2 등이 있다.
또 내부 타이머 간에 동기화를 위해서 사용하는 신호는 ITR0~ITR3 등이 있다.
Trigger 는 Master/Slave 설정에 따라서 Master 가 되면 다른 타이머로 Trigger 신호를 보내고, 이때 Slave 가 받게 되는 신호가 ITR0~ITR3 이다.

Master 가 TRGO1 이벤트를 보내면 Slave 가 ITR1 으로 해당 신호를 받아서 카운터를 동작시킨다.




3. 외부에서 들어는 TIMx_CH1,2,3,4 의 신호가 발생했을 때 Capture Compare 레지스터에 신호 발생시의 카운터 레지스터 값이 저장된다.
Capture compare 레지스터에 기록되는 값들을 계산해서 펄스의 파형을 계산해 낸다.

4. OCxREF 신호에 의해서 외부 핀으로 PWM 파형을 발생시키게 된다.

<Register Map>

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | **TIMx_CR1** | Reserved | | | | | | | | | | | | | | | | | | | | | | CKD [1:0] | | ARPE | CMS [1:0] | | DIR | OPM | URS | UDIS | CEN |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x04 | **TIMx_CR2** | Reserved | | | | | | | | | | | | | | | | | | | | | | | | TI1S | MMS[2:0] | | | CCDS | Reserved | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | | | |
| 0x08 | **TIMx_SMCR** | Reserved | | | | | | | | | | | | | | | | ETP | ECE | ETPS [1:0] | | ETF[3:0] | | | | MSM | TS[2:0] | | | Reserved | SMS[2:0] | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 |
| 0x0C | **TIMx_DIER** | Reserved | | | | | | | | | | | | | | | | | TDE | COMDE | CC4DE | CC3DE | CC2DE | CC1DE | UDE | Reserved | TIE | Reserved | CC4IE | CC3IE | CC2IE | CC1IE | UIE |
| | Reset value | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | | 0 | 0 | 0 | 0 | 0 |
| 0x10 | **TIMx_SR** | Reserved | | | | | | | | | | | | | | | | | | | CC4OF | CC3OF | CC2OF | CC1OF | Reserved | | TIF | Reserved | CC4IF | CC3IF | CC2IF | CC1IF | UIF |
| | Reset value | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | | | 0 | | 0 | 0 | 0 | 0 | 0 |
| 0x14 | **TIMx_EGR** | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | TG | Reserved | CC4G | CC3G | CC2G | CC1G | UG |
| | Reset value | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | | 0 | 0 | 0 | 0 | 0 |
| 0x18 | **TIMx_CCMR1** *Output Compare mode* | Reserved | | | | | | | | | | | | | | | | OC2CE | OC2M [2:0] | | | OC2PE | OC2FE | CC2S [1:0] | | OC1CE | OC1M [2:0] | | | OC1PE | OC1FE | CC1S [1:0] | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **TIMx_CCMR1** *Input Capture mode* | Reserved | | | | | | | | | | | | | | | | IC2F[3:0] | | | | IC2 PSC [1:0] | | CC2S [1:0] | | IC1F[3:0] | | | | IC1 PSC [1:0] | | CC1S [1:0] | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x1C | **TIMx_CCMR2** *Output Compare mode* | Reserved | | | | | | | | | | | | | | | | O24CE | OC4M [2:0] | | | OC4PE | OC4FE | CC4S [1:0] | | OC3CE | OC3M [2:0] | | | OC3PE | OC3FE | CC3S [1:0] | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **TIMx_CCMR2** *Input Capture mode* | Reserved | | | | | | | | | | | | | | | | IC4F[3:0] | | | | IC4 PSC [1:0] | | CC4S [1:0] | | IC3F[3:0] | | | | IC3 PSC [1:0] | | CC3S [1:0] | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x20 | **TIMx_CCER** | Reserved | | | | | | | | | | | | | | | | | | CC4P | CC4E | Reserved | | CC3P | CC3E | Reserved | | CC2P | CC2E | Reserved | | CC1P | CC1E |
| | Reset value | | | | | | | | | | | | | | | | | | | 0 | 0 | | | 0 | 0 | | | 0 | 0 | | | 0 | 0 |
| 0x24 | **TIMx_CNT** | Reserved | | | | | | | | | | | | | | | | CNT[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x28 | **TIMx_PSC** | Reserved | | | | | | | | | | | | | | | | PSC[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x2C | **TIMx_ARR** | Reserved | | | | | | | | | | | | | | | | ARR[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x30 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x34 | **TIMx_CCR1** | Reserved | | | | | | | | | | | | | | | | CCR1[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x38 | **TIMx_CCR2** | Reserved | | | | | | | | | | | | | | | | CCR2[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x3C | **TIMx_CCR3** | Reserved | | | | | | | | | | | | | | | | CCR3[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x40 | **TIMx_CCR4** | Reserved | | | | | | | | | | | | | | | | CCR4[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x44 | Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x48 | **TIMx_DCR** | Reserved | | | | | | | | | | | | | | | | | | | | DBL[4:0] | | | | Reserved | | | | DBA[4:0] | | | |
| | Reset value | | | | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | | | | | 0 | 0 | 0 | 0 | 0 |
| 0x4C | **TIMx_DMAR** | Reserved | | | | | | | | | | | | | | | | DMAB[15:0] | | | | | | | | | | | | | | | |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 4.10.2 TIM_TimeBaseInit

STM32 는 개별 Capture/Compare 채널을 가지고 있다.
Capture 와 Compare 기능에 대해서 간단히 설명하면 다음과 같다.

| | |
|---|---|
| Input Capture | 외부 신호 발생시 카운터 레지스터의 값을 Capture/Compare 레지스터에 저장 |
| Output Compare | 카운터 레지스터의 값과 Capture/Compare 레지스터의 값이 같을 경우 출력 이벤트 발생 |

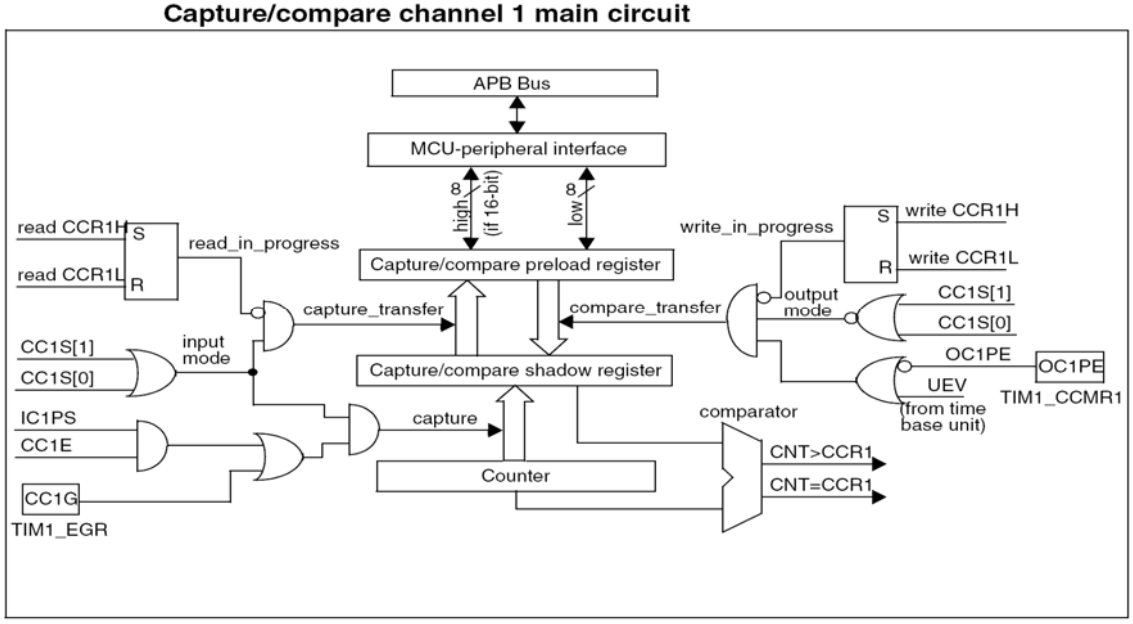
Capture 와 compare 는 Capture/Compare 레지스터를 공유해서 사용하기 때문에, 동시에 둘 다 사용할 수 없다.

<Capture/Compare Diagram>
Capture/Compare 의 기본 구조에 대해서 살펴보면 Capture/Compare 모드를 Capture 로 사용할지, Compare 로 사용할지는 CC1S[0:1]의 설정에 따라 결정이다.
Capture 모드로 설정되면 왼쪽 부분이며, Compare 모드로 설정되면 되면 오른쪽 부분이 동작한다.

<Input Capture>
먼저 Input Capture 부분을 보면 가운데 CC1S[0:1]의 값이 input mode 를 결정하는 값이다. 그림 아래 Capture 신호가 발생하는 경우는 CC1E (Capture/Compare Enable)가 SET 되어 있고, IC1PS(input Capture Prescaler Signal)의 신호가 발생한 경우이다.
또는 CC1G(Capture/Compare Generation)를 SW 적으로 SET 하여 capture 이벤트를 발생시킬 수 있다.
그림 아래 Capture 이벤트가 발생하면 Counter Register(CCR)의 값이 CCSR(Capture/Compare Shadow Register)로 복사이다.
CCSR 과 CCPR(Capture/Compare Preload Register)은 비슷하지만 조금 다른 레지스터이다. 메모리 버스에서는 CCPR 만 접근이 가능하며, capture_transfer, compare_transfer 신호에 의해서 CCSR 에 CCPR 의 값을 Write 하거나 CCSR 의 값을 CCPR 로 Write 할수 있다. CCPR 과 CCSR 은 CCR1H, CCR1L 의 신호에 의해서 8bit 씩 데이터를 교환한다.
여기서 중요한 것은 CCSR 에 직접 접근할수 없으며, CCPR 을 통해서 CCSR 에 접근한다는 것이다.

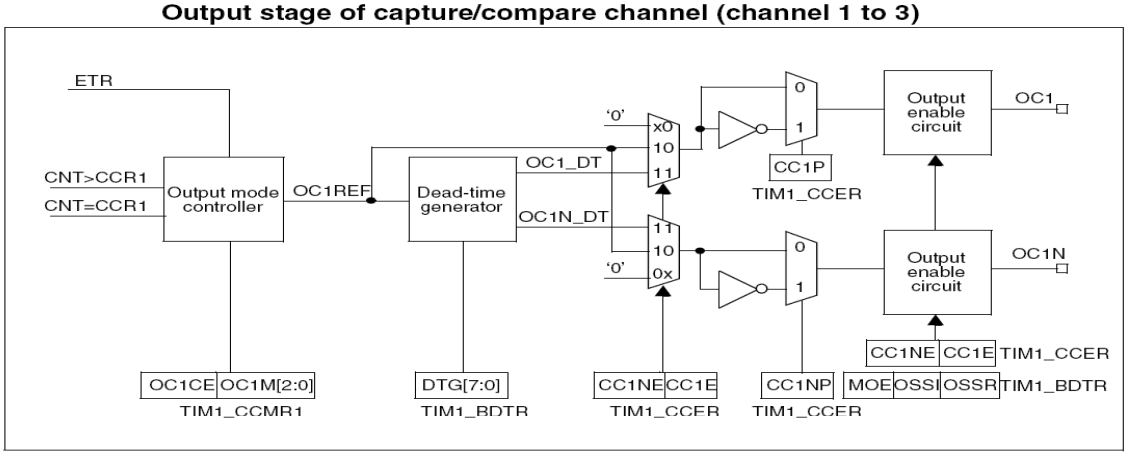
<Output Compare>
Output Compare 에서 아래 그림 왼쪽 Compare_transfer 가 발생하는 경우를 보면, UEV(Update Event)가 발생하거나, OC1PE 값이 Set 되지 않은 경우이다.
OC1PE(Output Compare Preload Enable)가 False 인 경우에는 CCPR 에 값을 적으면 CCSR 에 바로 적용이 되며, True 인 경우에는 CCPR 에 값을 적더라도 UEV 가 발생하지 않으면 CCSR 에 적용이 되지 않다.

**Capture/compare channel 1 main circuit**

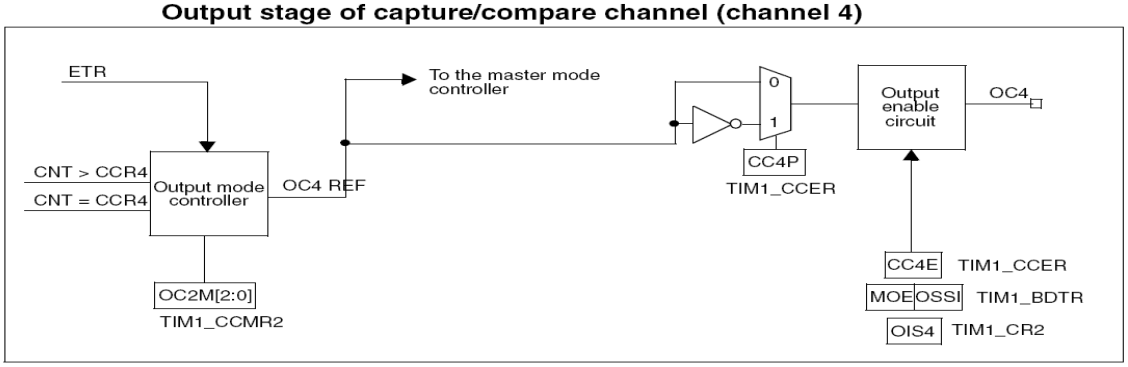



아래 그림 오른쪽을 보면 비교기는 CCSR 과 CCR 을 비교해서 CNT>CCR1, CNT=CCR1 신호를 보낸다.
해당 신호는 Output mode controller 를 통해서 출력 핀으로 연결이다.

**Output stage of capture/compare channel (channel 1 to 3)**




**Output stage of capture/compare channel (channel 4)**



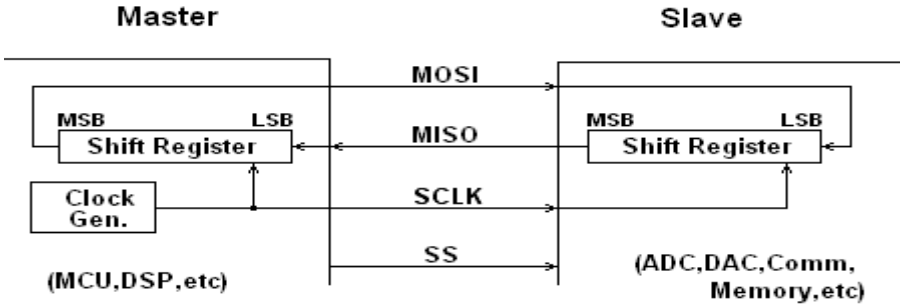
OC1M[2:0]에서는 입력 신호에 따라서 설정된 출력 신호 OC1REF 를 내보낸다.
OC1M 의 설정은 다음과 같다.
<OC Mode 설정표>

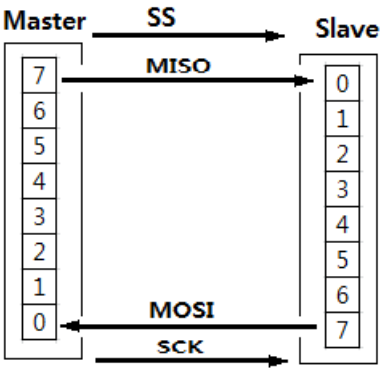
| | |
|---|---|
| 000 | OC1REF 신호 발생 안 함.<br>타이밍 인터럽트만 발생시킴. |
| 001 | CNT=CCR1 인 경우 OC1REF 가 high 로 설정됨. |
| 010 | CNT=CCR1 인 경우 OC1REF 가 low 로 설정됨. |
| 011 | CNT=CCR1 인 경우 OC1REF 가 toggle 됨. |
| 100 | OC1REF 가 low 로 됨. |
| 101 | OC1REF 가 high 로 됨. |
| 110 | PWM mode 1<br>업 카운팅 모드에서 CNT<CCR1 인 경우 OC1REF 는 high 로,<br>그렇지 않으면 low<br>다운 카운팅 모드에서 CNT>CCR1 인 경우 OC1REF 는 low,<br>그렇지 않으면 high |
| 111 | PWM mode 2<br>업 카운팅 모드에서 CNT<CCR1 인 경우 OC1REF 는 low,<br>그렇지 않으면 high<br>다운 카운팅 모드에서 CNT>CCR1 인 경우 OC1REF 는 high ,<br>그렇지 않으면 low |

OC1REF 의 신호는 CC1P(Capture/Compare Polarity) 설정에 따라서 출력 핀의 Active 설정이 변경된다.
CC1P 가 0 인 경우 active high, 1 인 경우 active low 이다.
즉, CC1P 가 1 인 경우 출력 신호가 반전되어 출력한다.

## 4.11 SPI

SPI(Serial Peripheral Interface) 통신은 4개의통신신호(/SS, SCK, MISO, MOSI)를 사용하여 클럭에 동기를 맞추어 통신을 한다.
마스터 모드와 슬레이브 모드로 설정 사용이 가능하고, 마스터장치는 클럭을 발생하여 송신을 한다. 2개의 데이터신호가 있으므로, 송신과 동시에 수신도 가능하며, 송수신 인터럽트 사용을 한다.
STM32에서는 8비트및 16비트모드를 사용하며, 최대전송속도는 PCLK의 2분주까지 한다. PCLK가 36MHz인 경우에 18MHz까지 동작한다. 통신클럭의 극성변경이 가능하고, MSB-first, LSB-first 조정이 가능하다. Hardware적인 CRC도 가능하고 멀티 마스터 모드도 사용한다.
다른 디바이스와 연결은 서로 같은 핀 이름으로 서로 연결한다.



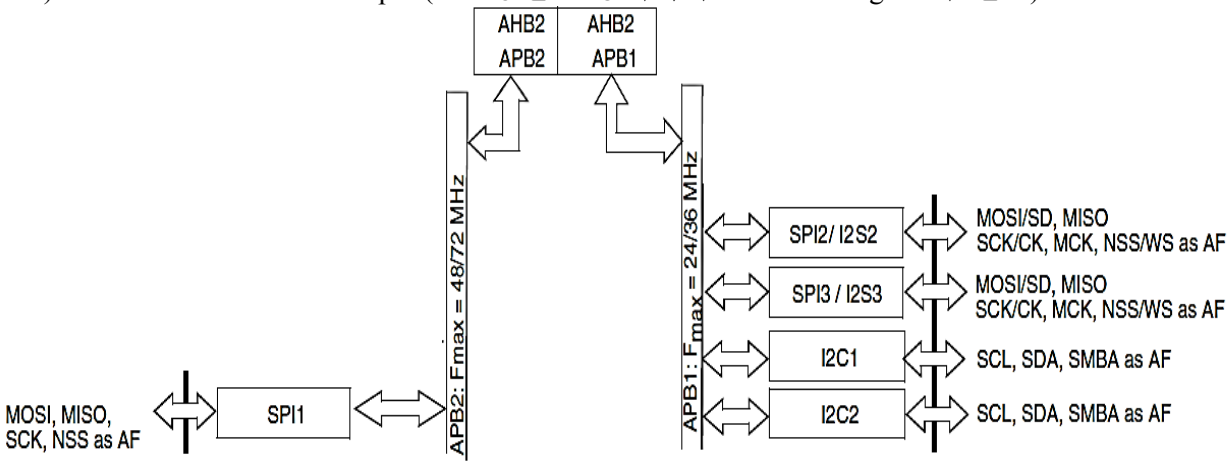

그림에서 알 수 있듯이 master 와 slave 모두본인의 MSB 부터 전송하며 받은 data 는 LSB 부터 채워 나감을 알 수 있다. 이때 동작구조를 선택할 수 있는데

1) master 만 slave 로 data 전송
   (master 가 data 전송, dummy 수신)
2) master 와 slave 간 상호전송
   (master 와 slave 의 data 가교환)
3) slave 만 master 로 data 전송
   (slave 만 data 전송, dummy 수신)

타 장비와 연결기본구성
SPI 는 4 개의 신호선으로 구성된다.

1) SPISOMI : SPI slave-output/master-input pin
   (slave 에서는 전송, master 에서는 수신을 하는 pin)
2) SPISIMO : SPI slave-input/master-output pin
   (slave 에서는 수신, master 에서는 전송을 하는 pin)

3) /SPISTE : SPI Slave transmit-enable pin
4) SPICLK : SPI serial-clock pin ( SPI 통신은 동기식이므로 CLK signal 이 필요)




STM32F103 의 SPI 내부구조

SPI1 는 APB2의 72MHz , SPI2,3 는 APB1의 36 MHz 로 동작한다.

STM32F103 의 SPI 핀

| LQFP64 | Pin name | Type | Main function (after reset) | Default | Remap |
|---|---|---|---|---|---|
| 20 | PA4 | I/O | PA4 | **SPI1_NSS**/ USART2_CK DAC_OUT1/ADC12_IN4 | |
| 21 | PA5 | I/O | PA5 | **SPI1_SCK** DAC_OUT2 ADC12_IN5 | |
| 22 | PA6 | I/O | PA6 | **SPI1_MISO** TIM8_BKIN/ADC12_IN6 TIM3_CH1 | |
| 23 | PA7 | I/O | PA7 | **SPI1_MOSI**/ TIM8_CH1N/ADC12_IN7 TIM3_CH2 | |
| 33 | PB12 | I/O | PB12 | **SPI2_NSS**/I2S2_WS/ I2C2_SMBA/USART3_CK/ TIM1_BKIN | |
| 34 | PB13 | I/O | PB13 | **SPI2_SCK**/I2S2_CK USART3_CTS/TIM1_CH1N | |
| 35 | PB14 | I/O | PB14 | **SPI2_MISO**/TIM1_CH2N USART3_RTS/ | |
| 36 | PB15 | I/O | PB15 | **SPI2_MOSI**/I2S2_SD TIM1_CH3N/ | |
| 50 | PA15 | I/O | JTDI | **SPI3_NSS**/ I2S3_WS | TIM2_CH1_ETR PA15 / SPI1_NSS |
| 55 | PB3 | I/O | JTDO | **SPI3_SCK** / I2S3_CK/ | PB3/TRACESWO TIM2_CH2 / SPI1_SCK |
| 56 | PB4 | I/O | NJTRST | **SPI3_MISO** | PB4 / TIM3_CH1 SPI1_MISO |
| 57 | PB5 | I/O | PB5 | I2C1_SMBA/ **SPI3_MOSI** /I2S3_SD | TIM3_CH2 / SPI1_MOSI |

여기서 사용하는 디바이스에는 SPI 는 3개를 사용할 수 있다.

**SPI characteristics**

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| $f_{SCK}$<br>$1/t_{c(SCK)}$ | SPI clock frequency | Master mode | | 18 | MHz |
| | | Slave mode | | 18 | |
| $t_{r(SCK)}$<br>$t_{f(SCK)}$ | SPI clock rise and fall time | Capacitive load: C = 30 pF | | 8 | ns |
| DuCy(SCK) | SPI slave input clock duty cycle | Slave mode | 30 | 70 | % |
| $t_{su(NSS)}$ (1) | NSS setup time | Slave mode | $4t_{PCLK}$ | | ns |
| $t_{h(NSS)}$ (1) | NSS hold time | Slave mode | $2t_{PCLK}$ | | |
| $t_{w(SCKH)}$ (1)<br>$t_{w(SCKL)}$ (1) | SCK high and low time | Master mode, $f_{PCLK}$ = 36 MHz, presc = 4 | 50 | 60 | |
| $t_{su(MI)}$ (1)<br>$t_{su(SI)}$ (1) | Data input setup time | Master mode | 5 | | |
| | | Slave mode | 5 | | |
| $t_{h(MI)}$ (1) | Data input hold time | Master mode | 5 | | |
| $t_{h(SI)}$ (1) | | Slave mode | 4 | | |
| $t_{a(SO)}$ (1)(2) | Data output access time | Slave mode, $f_{PCLK}$ = 20 MHz | 0 | $3t_{PCLK}$ | |
| $t_{dis(SO)}$ (1)(3) | Data output disable time | Slave mode | 2 | 10 | |
| $t_{v(SO)}$ (1) | Data output valid time | Slave mode (after enable edge) | | 25 | |
| $t_{v(MO)}$ (1) | Data output valid time | Master mode (after enable edge) | | 5 | |
| $t_{h(SO)}$ (1) | Data output hold time | Slave mode (after enable edge) | 15 | | |
| $t_{h(MO)}$ (1) | | Master mode (after enable edge) | 2 | | |

프로그램작성

```
SPI_Cmd(SPI2, DISABLE);
//configure spi
SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_16;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_Init(SPI2, &SPI_InitStructure);
SPI_Cmd(SPI2, ENABLE);
```

1) SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
2 line 으로 양방향 통신을 한다.

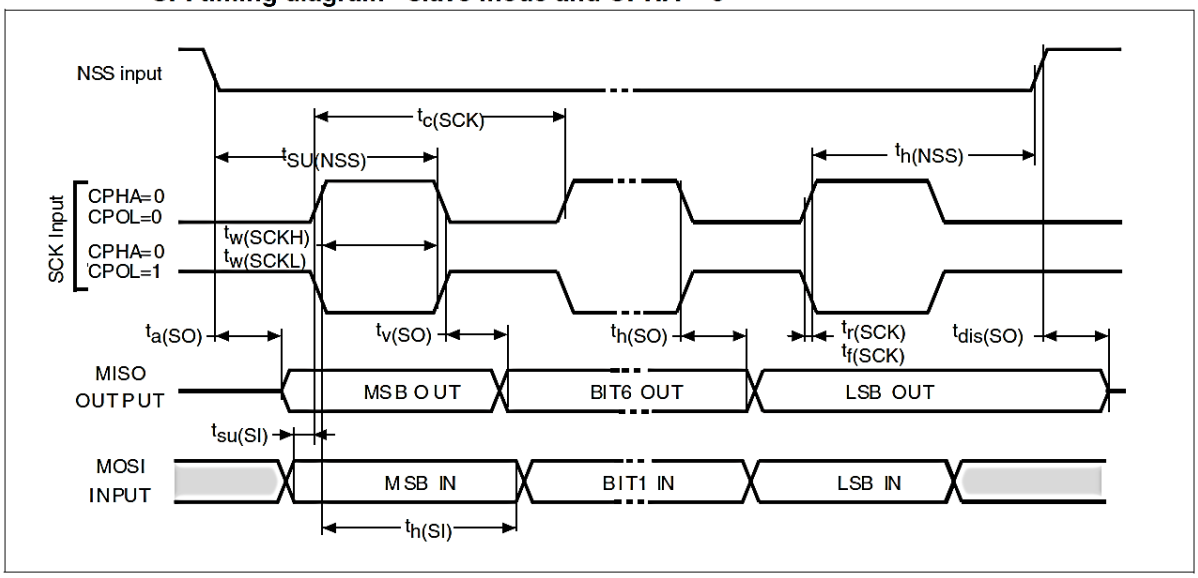
2) SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
Mode를 Master로 설정한다.

3) SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;

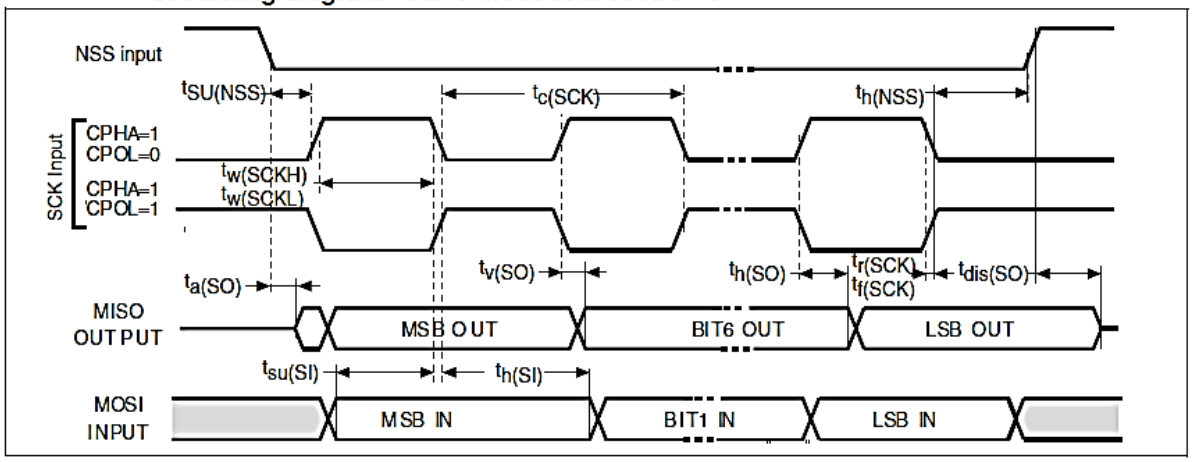
Data Size로 8bit 또는10bit 방식을 정하는데 여기서는 8bit 방식이다.

4) SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
   SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;

**SPI timing diagram - slave mode and CPHA = 0**




**SPI timing diagram - slave mode and CPHA = 1**




위의 타이밍을 정리하면 다음과 같다.

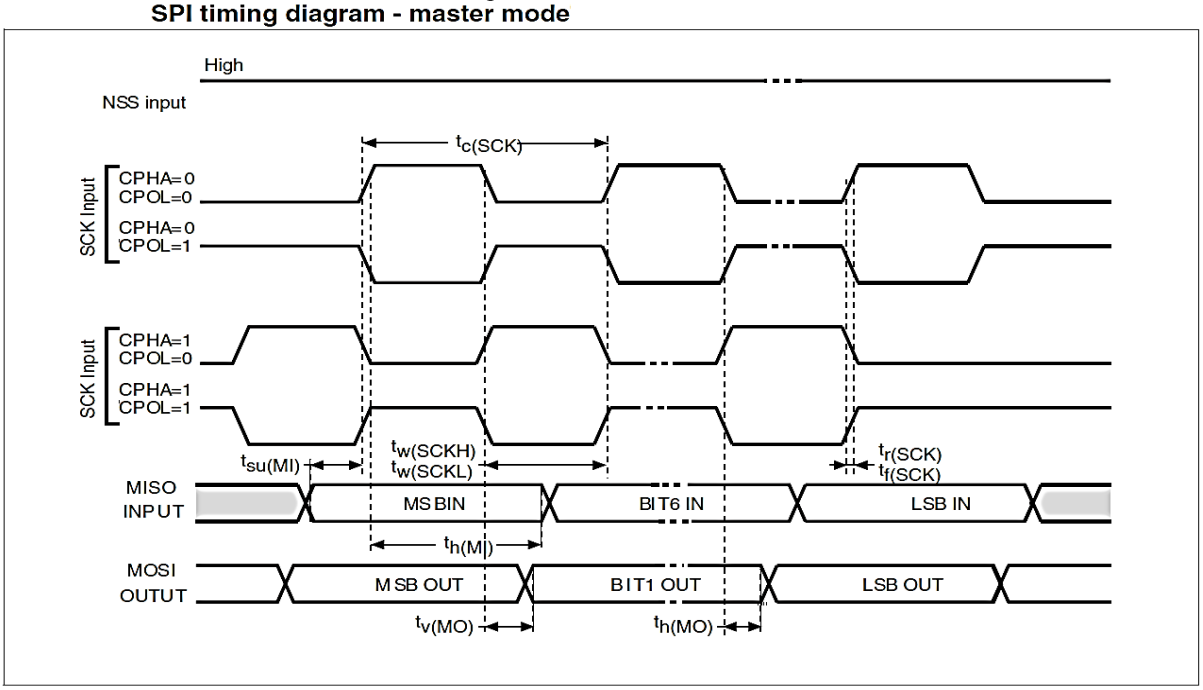
CPOL and CPHA Functionality

| | Leading edge | Trailing edge |
|---|---|---|
| CPOL = 0, CPHA = 0 | Sample (Rising) | Setup (Falling) |
| CPOL = 0, CPHA = 1 | Setup (Rising) | Sample (Falling) |
| CPOL = 1, CPHA = 0 | Sample (Falling) | Setup (Rising) |
| CPOL = 1, CPHA = 1 | Setup (Falling) | Sample (Rising) |

SPI와 연결하고자 하는 디바이스 스펙을 보면 위의 4가지 중 한가지 방법으로 되어있다.
SCK Input에 따라 CPOL, CPHA의 0, 1을 결정한다.

Master 모드 일 때도 마찬가지로 SCK Input에 따라 CPOL, CPHA의 0, 1을 결정한다.

**SPI timing diagram - master mode**




5)SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
Slave와 통신을 하기 위해서는 Start 는 0으로 End 는 1로 한다.

6)SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_16;

SPI_BaudRatePrescaler를 정한다.

7)SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
Data를 전송할 때 순서가 MSB First Bit로 설정한다.

8)SPI_InitStructure.SPI_CRCPolynomial = 7;
CRC 설정하는 부분으로 초기값을 설정한다. CRC 에 대한 계산식이 내장되어있다.

9)SPI_Init(SPI2, &SPI_InitStructure);
SPI 설정한 값을 저장한다. 여기서는 SPI2를 설정한다.

10)SPI_Cmd(SPI2, ENABLE);
SPI를 시작한다. 여기서는 SPI2를 설정한다.

```
static void SPI_Rcc_Configuration(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO,ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2,ENABLE);
}

static void GPIO_Configuration(void)
{
	GPIO_InitTypeDef GPIO_InitStruct;

	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
	GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
	GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;

	GPIO_Init(GPIOB, &GPIO_InitStruct);
}

void SPI_Configuration(void)
{
	SPI_InitTypeDef SPI_InitStruct;
	SPI_Rcc_Configuration();
	SPI_InitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;

       // 2라인양방향통신
	SPI_InitStruct.SPI_Direction= SPI_Direction_2Lines_FullDuplex;
	마스터설정
	SPI_InitStruct.SPI_Mode = SPI_Mode_Master;
	8bit  설정
	SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;

	SPI_InitStruct.SPI_CPOL = SPI_CPOL_High;
	SPI_InitStruct.SPI_CPHA = SPI_CPHA_2Edge;
	SPI_InitStruct.SPI_NSS = SPI_NSS_Soft;

	SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
	SPI_InitStruct.SPI_CRCPolynomial = 7;
	SPI_Init(SPI2, &SPI_InitStruct);
	GPIO_Configuration();
	SPI_Cmd(SPI2, ENABLE);
}
void SPI2_Send_byte(u16 data)
{
	while(SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE)==RESET);
	SPI_I2S_SendData(SPI2,data);
	while(SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE)==RESET);
	SPI_I2S_ReceiveData(SPI2);
}
u16 SPI2_Receive_byte(void)
{
	while(SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE)==RESET);
	SPI_I2S_SendData(SPI2,0x00);

	while(SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE)==RESET);
	return SPI_I2S_ReceiveData(SPI2);
}
```

## 4.12 I2C

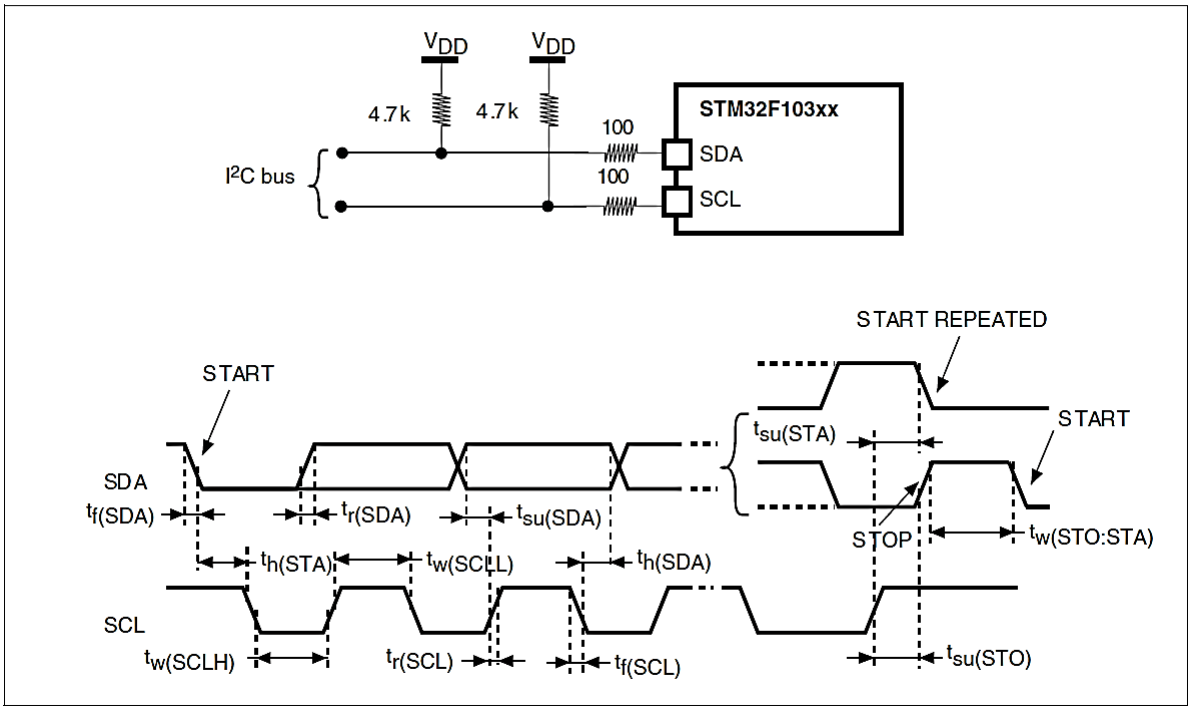
I²C 는 필립스에서 개발한 직렬 컴퓨터버스이며 마더보드, 임베디드 시스템, 휴대폰등에 저속의 주변기기를 연결하기 위해 사용된다. I²C 라는 이름은 Inter-Integrated Circuit 의 약자이다. I2C 의 장점은 단지 2 개의 wire 로만 통신이 가능하다. 신호는 크게 START bit , DATA bits, STOP bit 의 3 가지로 나뉜다. START 와 STOP 은 SCL 이 HIGH 일 때 변하며, DATA 는 SCL 이 LOW 일 때 변한다. (protocol 이 진행 중이지 않을 경우에는 pull-up register 에 의해 SDA, SCL 모두 HIGH 상태이다.)
START 는 SCL 이 HIGH 일 때 SDA 가 Falling 하는 경우로 slave 에게 protocol 이 시작됨을 알린다.
STOP 은 SCL 이 HIGH 일 때 SDA 가 Rising 하는 경우로 slave 에게 protocol 이 종료됨을 알린다.
DATA 는 SCL 이 LOW 일 때 SDA 가변하며 SCL 이 HIGH 일 때 SDA 값이 valid 하다.
또한 DATA bits 는 ACK 와 R/W 신호를 포함한다.(R/W 신호는 HIGH 일 경우 READ, LOW 일 경우 WRITE 이다.)
I2C 전송은 마스터입장에서 크게 2 가지 경우로 나뉜다. Slave 에 data 를 write 하는 Transfer 와 Slave 로부터 read 하는 receive 의 경우이다.
START 와 STOP, R/W 신호는 무조건 Master 에서 생성하는 것이나 data 와 ack 는 Transfer 인지 Receive 인지에 따라 Master 혹은 Slave 에서 생성한다.

**I²C characteristics**

| Symbol | Parameter | Standard mode I²C(1) | | Fast mode I²C(1)(2) | | Unit |
|---|---|---|---|---|---|---|
| | | Min | Max | Min | Max | |
| $t_{w(SCLL)}$ | SCL clock low time | 4.7 | | 1.3 | | µs |
| $t_{w(SCLH)}$ | SCL clock high time | 4.0 | | 0.6 | | |
| $t_{su(SDA)}$ | SDA setup time | 250 | | 100 | | ns |
| $t_{h(SDA)}$ | SDA data hold time | 0(3) | | 0(4) | 900(3) | |
| $t_{r(SDA)}$ $t_{r(SCL)}$ | SDA and SCL rise time | | 1000 | 20 + 0.1$C_b$ | 300 | |
| $t_{f(SDA)}$ $t_{f(SCL)}$ | SDA and SCL fall time | | 300 | | 300 | |
| $t_{h(STA)}$ | Start condition hold time | 4.0 | | 0.6 | | µs |
| $t_{su(STA)}$ | Repeated Start condition setup time | 4.7 | | 0.6 | | |
| $t_{su(STO)}$ | Stop condition setup time | 4.0 | | 0.6 | | µs |
| $t_{w(STO:STA)}$ | Stop to Start condition time (bus free) | 4.7 | | 1.3 | | µs |
| $C_b$ | Capacitive load for each bus line | | 400 | | 400 | pF |

위의 SPI의 그림 내부에 있는 것처럼 클럭은 I2C1,2 는 APB1의 36 MHz 로 동작한다.

**I²C bus AC waveforms and measurement circuit**




| fSCL (kHz) | I2C_CCR valu<br>RP = 4.7 kΩ |
|---|---|
| 400 | 0x801E |
| 300 | 0x8028 |
| 200 | 0x803C |
| 100 | 0x00B4 |
| 50 | 0x0168 |
| 20 | 0x0384 |

**SCL frequency (fPCLK1= 36 MHz.,VDD = 3.3 V)**

| LQFP64 | Pin name | Type | Main function (after reset) | Default | Remap |
|---|---|---|---|---|---|
| 58 | PB6 | I/O | PB6 | **I2C1_SCL**/ TIM4_CH1 | USART1_TX |
| 59 | PB7 | I/O | PB7 | **I2C1_SDA** / FSMC_NADV /TIM4_CH2 | USART1_RX |
| 61 | PB8 | I/O | PB8 | TIM4_CH3/SDIO_D4 | I2C1_SCL/ CAN_RX |
| 62 | PB9 | I/O | PB9 | TIM4_CH4/SDIO_D5 | I2C1_SDA / CAN_TX |
| 29 | PB10 | I/O | PB10 | **I2C2_SCL**/USART3_TX | TIM2_CH3 |
| 30 | PB11 | I/O | PB11 | **I2C2_SDA**/USART3_RX | TIM2_CH4 |

여기서 사용하는 디바이스는 I2C 가 2개를 사용한다.
PB6 I2C1_SCL, PB7 I2C1_SDA
PB10 I2C2_SCL, PB11 I2C2_SDA
참고로 PB8(I2C1_SCL), PB9(I2C1_SDA) Remap 하여 사용할 수 있다. (이때 동시에 PB6,7 은 사용불가)

다음아래는 24c02 프로그램설정 예제이다.

```
void I2C_Configuration(void)
{
    I2C_InitTypeDef    I2C_InitStructure;
    GPIO_InitTypeDef    GPIO_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO , ENABLE);

    /* Configure I2C1 pins: PB6->SCL and PB7->SDA */
    GPIO_InitStructure.GPIO_Pin =    GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    I2C_DeInit(I2C1);
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    I2C_InitStructure.I2C_OwnAddress1 = 0x30;
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_ClockSpeed = 100000;

    I2C_Cmd(I2C1, ENABLE);
    I2C_Init(I2C1, &I2C_InitStructure);

    I2C_AcknowledgeConfig(I2C1, ENABLE);
}

static void I2C_AcknowledgePolling(I2C_TypeDef *I2Cx,uint8_t I2C_Addr)
{
  vu16 SR1_Tmp;
  do
  {
    I2C_GenerateSTART(I2Cx, ENABLE);

    SR1_Tmp = I2C_ReadRegister(I2Cx, I2C_Register_SR1);

#ifdef AT24C01A

        I2C_Send7bitAddress(I2Cx, I2C_Addr, I2C_Direction_Transmitter);
#else

        I2C_Send7bitAddress(I2Cx, 0, I2C_Direction_Transmitter);
#endif

  }while(!(I2C_ReadRegister(I2Cx, I2C_Register_SR1) & 0x0002));

  I2C_ClearFlag(I2Cx, I2C_FLAG_AF);

  I2C_GenerateSTOP(I2Cx, ENABLE);
}


uint8_t I2C_Read(I2C_TypeDef *I2Cx,uint8_t I2C_Addr,uint8_t addr,uint8_t *buf,uint16_t num)
{
```

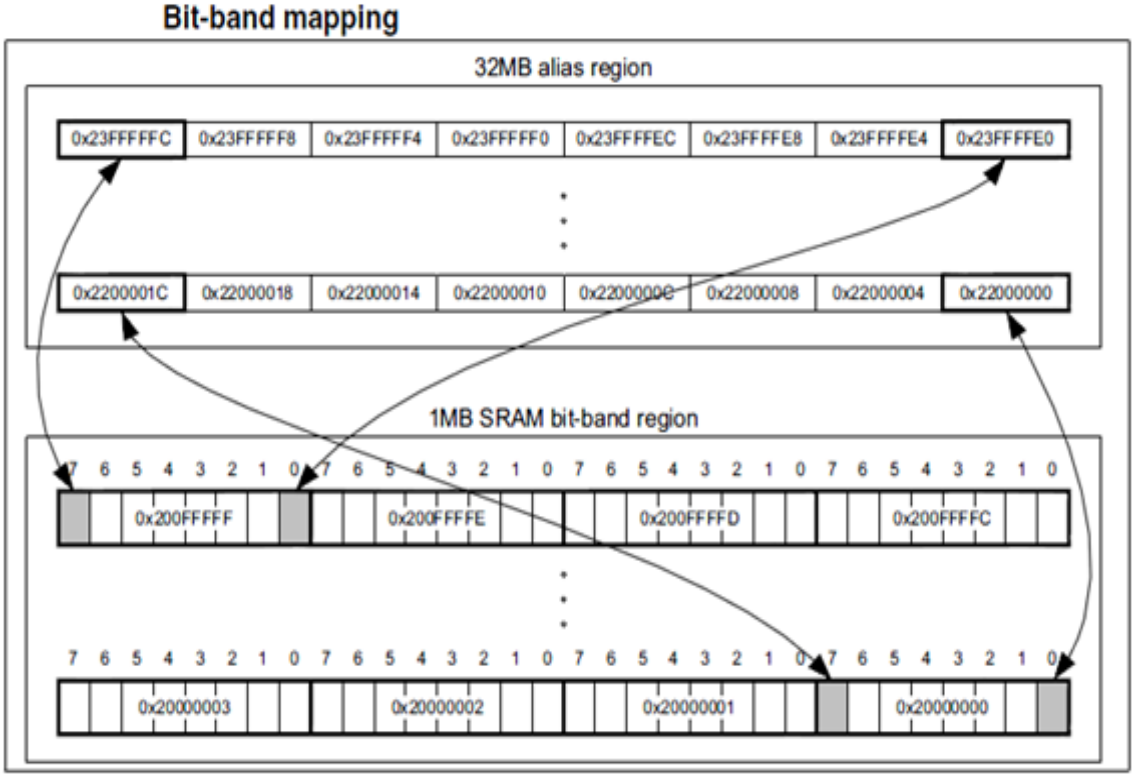
```
    if(num==0)
        return 1;

        while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));

        I2C_AcknowledgeConfig(I2Cx, ENABLE);

    I2C_GenerateSTART(I2Cx, ENABLE);
    while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

#ifdef AT24C01A
    I2C_Send7bitAddress(I2Cx,   I2C_Addr, I2C_Direction_Transmitter);
    while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

        I2C_SendData(I2Cx, addr);
    while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTART(I2Cx, ENABLE);
        while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

        I2C_Send7bitAddress(I2Cx, I2C_Addr, I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

#else
        I2C_Send7bitAddress(I2Cx, addr<<1, I2C_Direction_Receiver);
        while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
#endif

    while (num)
    {
                if(num==1)
                {
                I2C_AcknowledgeConfig(I2Cx, DISABLE);
                I2C_GenerateSTOP(I2Cx, ENABLE);
                }

                while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED));   /*
EV7 */
            *buf = I2C_ReceiveData(I2Cx);
            buf++;
            /* Decrement the read bytes counter */
            num--;
    }
        I2C_AcknowledgeConfig(I2Cx, ENABLE);
        return 0;
}

uint8_t I2C_WriteOneByte(I2C_TypeDef *I2Cx,uint8_t I2C_Addr,uint8_t addr,uint8_t value)
{
        I2C_GenerateSTART(I2Cx, ENABLE);
        while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

#ifdef AT24C01A
        I2C_Send7bitAddress(I2Cx, I2C_Addr, I2C_Direction_Transmitter);
        while(!I2C_CheckEvent(I2Cx,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
```

```
		I2C_SendData(I2Cx, addr);
		while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

#else
		I2C_Send7bitAddress(I2Cx, addr<<1, I2C_Direction_Transmitter);
		while(!I2C_CheckEvent(I2Cx,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
#endif

		I2C_SendData(I2Cx, value);
		while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

		I2C_GenerateSTOP(I2Cx, ENABLE);
		I2C_AcknowledgePolling(I2Cx,I2C_Addr);
		I2C_delay(1000);
		return 0;
}

uint8_t I2C_Write(I2C_TypeDef *I2Cx,uint8_t I2C_Addr,uint8_t addr,uint8_t *buf,uint16_t num)
{
		uint8_t err=0;

		while(num--)
		{
			if(I2C_WriteOneByte(I2Cx, I2C_Addr,addr++,*buf++))
			{
				err++;
			}
		}
		if(err)
			return 1;
		else
			return 0;
}
```

## 4.13 참고

### 4.13.1 Bit Banding

Bit Banding 은 bit operation 을 빠르게 해 준다. 특정 영역의 비트를 어드레스에 맵핑해서 제어하는 기능이다.
다음 그림과 같이 하나의 비트가 하나의 주소에 맵핑이다.




1 비트당 32Bit 단위 주소에 맵핑되기 때문에 1MB 의 영역이 총 32MB 의 주소 영역에 맵핑이다. 맵핑된 주소에서 Bit[0]만 사용되며, Bit[31:1]은 사용되지 않다.
이런 Bit Banding 이 지원되는 Bit Band Region 은 SRAM 영역, Peripheral 영역에서 각각 1MB 씩 지원이다.

**SRAM memory bit-banding regions**

| Address range | Memory region | Instruction and data accesses |
|---|---|---|
| 0x20000000-0x200FFFFF | SRAM bit-band region | Direct accesses to this memory range behave as SRAM memory accesses, but this region is also bit addressable through bit-band alias. |
| 0x22000000-0x23FFFFFF | SRAM bit-band alias | Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not remapped. |

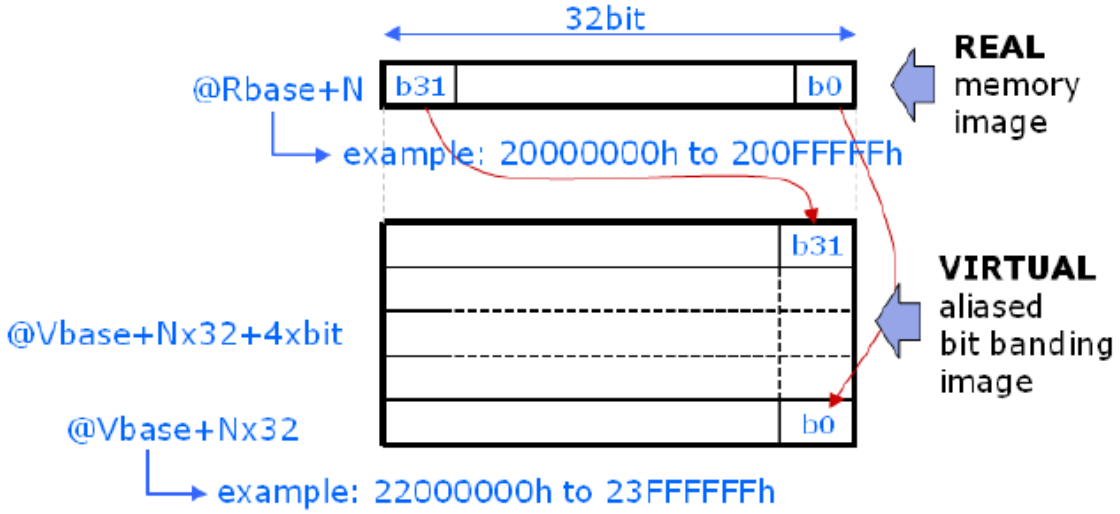
**Peripheral memory bit-banding regions**

| Address range | Memory region | Instruction and data accesses |
|---|---|---|
| 0x40000000-0x400FFFFF | Peripheral bit-band region | Direct accesses to this memory range behave as peripheral memory accesses, but this region is also bit addressable through bit-band alias. |
| 0x42000000-0x43FFFFFF | Peripheral bit-band alias | Data accesses to this region are remapped to bit-band region. A write operation is performed as read-modify-write. Instruction accesses are not permitted. |

Bit Banding 의 장점.

1. Bit 제어의 효율성
   - Bit 제어에 있어서 코드 사이즈가 줄어들 수 있다.
2. 코드의 안정성
   - 제어하고자 하는 비트만 제어하므로, 다른 비트들에 영향을 주는 일이 없다.

<Bit Banding Address 공식>
Real Address Base 주소에서 N Offset 에 위치한 경우에 bit 번째 비트의 맵핑 어드레스는
Virtual Address Base + N*32 + 4*bit 이 이다.




<예제 코드 1>
RAM 영역의 데이터를 Bit Banding 을 통해서 알아본다.

```
#include "stm32f10x.h"
#include <stdio.h>
#define RAM_BASE            0x20000000
#define RAM_BB_BASE         0x22000000

#define   Var_ResetBit_BB(VarAddr, BitNumber)       \
          (*(__IO uint32_t *) (RAM_BB_BASE | ((VarAddr - RAM_BASE) << 5) |
((BitNumber) << 2)) = 0)
#define Var_SetBit_BB(VarAddr, BitNumber)           \
          (*(__IO uint32_t *) (RAM_BB_BASE | ((VarAddr - RAM_BASE) << 5) |
((BitNumber) << 2)) = 1)
#define Var_GetBit_BB(VarAddr, BitNumber)           \
          (*(__IO uint32_t *) (RAM_BB_BASE | ((VarAddr - RAM_BASE) << 5) |
((BitNumber) << 2)))

int ExtInt = 0x00000000;

int main()
{
```

```
    int StackInt = 0xffffffff;

        init_UART();

        printf("ExtInt Address : %X    Value: %X\r\n", &ExtInt, ExtInt);
    printf("StackInt Address : %X    Value: %X\r\n", &StackInt, StackInt);

        Var_SetBit_BB((int)&ExtInt, 0);
    Var_SetBit_BB((int)&ExtInt, 1);
    Var_SetBit_BB((int)&ExtInt, 2);
    Var_SetBit_BB((int)&ExtInt, 3);

    Var_ResetBit_BB((int)&StackInt, 0);
    Var_ResetBit_BB((int)&StackInt, 1);
    Var_ResetBit_BB((int)&StackInt, 2);
    Var_ResetBit_BB((int)&StackInt, 3);

    printf("ExtInt Address : %X    Value: %X\r\n", &ExtInt, ExtInt);
    printf("StackInt Address : %X    Value: %X\r\n", &StackInt, StackInt);

        return 0;
}
```

전역 변수 ExtInt 와 지역 변수 Stackint 를 Bit Bandig 하는 예제이다.
전역 변수의 하위 4 비트는 SET 하고, 지역 변수의 하위 4 비트는 RESET 한다.

<예제 코드 1 : 실행 결과>

```
ExtInt Address : 20000414    Value: 0
StackInt Address : 200003E0    Value: FFFFFFFF
ExtInt Address : 20000414    Value: F
StackInt Address : 200003E0    Value: FFFFFFF0
```

전역 변수와 지역 변수는 메모리 상에 할당되는 위치가 다릅니다.
지역 변수인 StackInt 는 Stack 내에 할당되었으며, 전역 변수는 Stack 외부에 할당되었다.
기억하시겠지만, 사용자는 Linker 설정에서 Stack 의 사이즈를 0x400 으로 직접 설정했었다.
Map 파일을 통해서 Stack 과 전역변수 Extint 가 할당된 위치를 보도록 하겠다.

<예제 코드 1 : Map 파일>

```
… 중략
CSTACK              uninit     0x20000000    0x400    <Block tail>
                                          - 0x20000400    0x400
… 중략
ExtInt                         0x20000414       0x4    Data    Gb    main.o [1]
...
```

Map 파일을 보시면 CSTACK 은 0x20000000~0x20000400 까지 사용하며,
실행 결과에서 보면 StackInt 는 0x200003E0 의 Stack 내에 위치하고 있다.
ExtInt 는 Map 파일에서 보듯이 0x20000414 에 할당되어 있다.

<코드 따라하기>

1. Var_ResetBit_BB, Var_SetBit_BB, Var_GetBit_BB
   - 위 Macro 는 변수의 어드레스와 Bit 번호를 입력 받아서 Bit Banding 하는 Macro 이다.
     만약 Peripheral 영역에 대해서도 동일한 Macro 를 만든다면, 아래 Peripheral Base 로 변경하면 이다.
     #define PERIPH_BB_BASE ((uint32_t)0x42000000)
     #define PERIPH_BASE ((uint32_t)0x40000000)
2. ExtInt 전역 변수
   - ExtInt 전역 변수의 초기값은 0x00000000 이었다.
     Var_SetBit_BB Macro 를 사용해서 하위 4 비트를 각각 Set 하였다.
3. StackInt 지역 변수
   - StackInt 지역 변수의 초기값은 0xFFFFFFFF 이었다.
     Var_ResetBit_BB Macro 를 사용해서 하위 4 비트를 각각 ReSet 하였다.

실제로 Bit Banding 은 SRAM 영역보다는 Peripheral 영역의 Register 들을 제어하는데 더 유용하게 사용될 것이다.

### 4.13.2 assert_failed

```
#ifdef   USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
{
   /* User can add his own implementation to report the file name and line number,
        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

   /* Infinite loop */
   while (1)
   {
   }
}
#endif
```

void assert_failed(uint8_t* file, uint32_t line)
assert_failed()에서 file 이 문장이 수행되는 함수가 존재하는 파일의 이름을 디렉토리까지 포함해서 내포하고, line은 그 파일의 line 의 수를 지니고 있다. 즉 파일이름과 라인 번호를 출력하고, 이 함수를 끝낸다.

### 4.13.3 GPIO JTAG

JTAG PORT 일반 GPIO 로 사용할 때 사용한다.

핀 번호

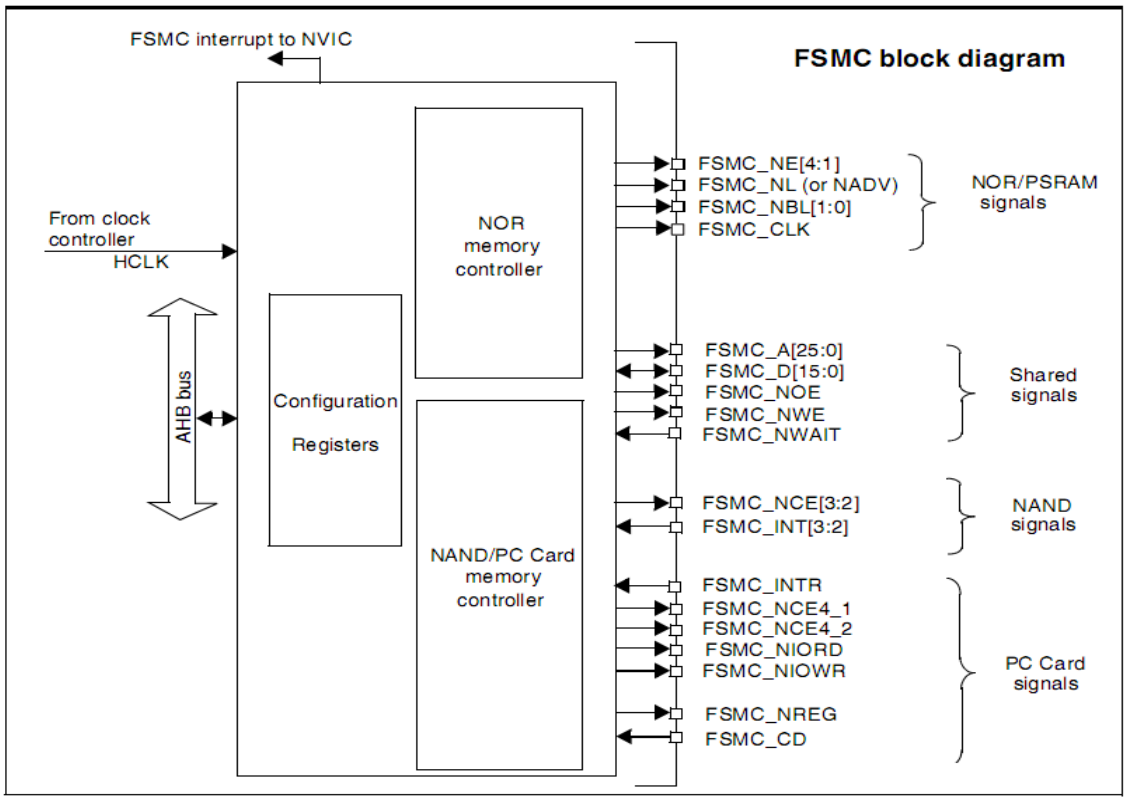
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| VCC | TRST | TDI | TMS | TCK | GND | TDO | /RST |
| VCC | PB4 | PA15 | PA13 | PA14 | GND | PB3 | /RST |

프로그램

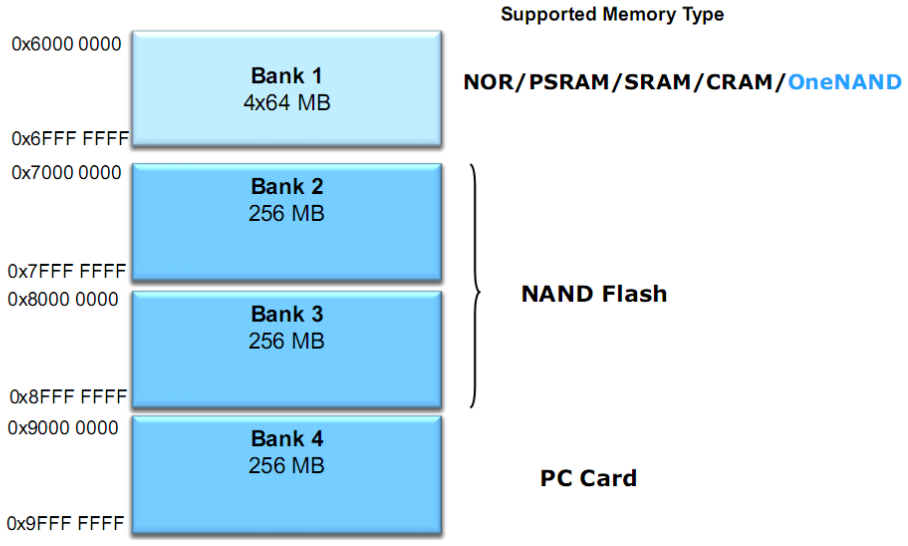
```
  // JTAG I/O 출력 으로 만들기     Enable GPIOA, GPIOB and AFIO clocks
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
RCC_APB2Periph_AFIO, ENABLE );

   //Disable the serial Wire Debug Port SWJ-DP
     GPIO_PinRemapConfig(GPIO_Remap_SWJ_Disable, ENABLE);

   // JTAG => I/O PA13, 14, 15
     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 |GPIO_Pin_14 |GPIO_Pin_15;
     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
     GPIO_Init(GPIOA, &GPIO_InitStructure);

   // JTAG => I/O PB3, 4
     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 |GPIO_Pin_4;
     GPIO_Init(GPIOB, &GPIO_InitStructure);
```

### 4.13.3 FSMC

FSMC 는 Flexible Static Memory Controller 의 약어이다. 이를 사용하여 외부에 메모리나 I/O 를 확장할 때 사용한다.




- For the FSMC, the external memory is divided into 4 fixed size banks of 4x64 MB each:
  - Bank 1 can be used to address NOR Flash, OneNAND or PSRAM memory devices.
  - Banks 2 and 3 can be used to address NAND Flash devices.
  - Bank 4 can be used to address a PC Card device.



BANK 1 에서는 TFTLCD, RAM, NOR 관련된 디바이스를 연결해서 사용한다.

1. TFTLCD를 연결할 때

PD7=NE1 (CE2) LCD
#define Bank1_LCD_C ((uint32_t)0x60000000) // RS =0 disp Reg ADDR
#define Bank1_LCD_D ((uint32_t)0x60020000) // RS = 1 disp Data ADDR

| 기능 | STM32F4 핀 번호 | STM32F4 핀 이름 |
|---|---|---|
| D0 | 61 | PD14 |
| D1 | 62 | PD15 |
| D2 | 81 | PD0 |
| D3 | 82 | PD1 |
| D4 | 38 | PE7 |
| D5 | 39 | PE8 |
| D6 | 40 | PE9 |
| D7 | 41 | PE10 |
| D8 | 42 | PE11 |
| D9 | 43 | PE12 |
| D10 | 44 | PE13 |
| D11 | 45 | PE14 |
| D12 | 46 | PE15 |
| D13 | 55 | PD8 |
| D14 | 56 | PD9 |
| D15 | 57 | PD10 |
| CS | 88 | PD7 (NE1) |
| RS | 60 | PD11 (FSMC-A16) |
| RD | 85 | PD4 () |
| WR | 86 | PD5 (NWE) |
| RESET(TFT_RST) | 25 (7) | 3 V (GPIO PC13) |
| TP_IRQ | 44 | PC4 |
| TP_SCK | 41 | PA5 |
| TP_SI | 43 | PA7 |
| TP_SO | 42 | PA6 |
| TP_CS | 40 | PA4 |
| BL_CNTL | 45 | PC5 |

2. NOR, RAM을 연결할 때

TFTLCD, NOR, RAM 서로 같지 않는 핀을 사용해야 한다. 예를 들어 TFTLCD 에서 PD7을 사용시 다른 PG9, PG10, PG12 중 하나를 선택해야 한다.

#define Bank1_SRAM2_ADDR ((uint32_t)0x60000000) 에서 Bank1_SRAM2_ADDR 는 사용자가 임의로 지정한 것이며, ((uint32_t)0x60000000) 는 어드레스를 나타낸 것이다. NE1에서 NE4까지 사용 가능하다.

#define FSMC_Bank1_NORSRAM1 ((uint32_t)0x00000000) 는 FSMC_Bank1_NORSRAM1 stm32f4xx_fsmc.h 에서 설정되어 있다.

PD7=NE1
#define Bank1_SRAM2_ADDR ((uint32_t)0x60000000)
#define FSMC_Bank1_NORSRAM1 ((uint32_t)0x00000000)

PG9=NE2
#define Bank1_SRAM2_ADDR ((uint32_t)0x64000000)
#define FSMC_Bank1_NORSRAM2 ((uint32_t)0x00000002)

PG10=NE3
#define Bank1_SRAM2_ADDR ((uint32_t)0x68000000)
#define FSMC_Bank1_NORSRAM3 ((uint32_t)0x00000004)

PG12=NE4
#define Bank1_SRAM2_ADDR ((uint32_t)0x6C000000)
#define FSMC_Bank1_NORSRAM4 ((uint32_t)0x00000006)

BANK2 에서는 NAND 를 연결할 때 사용한다.
PD7= CE2 NAND
#define Bank1_LCD_C ((uint32_t)0x70000000)
#define FSMC_Bank2_NAND ((uint32_t)0x00000010)

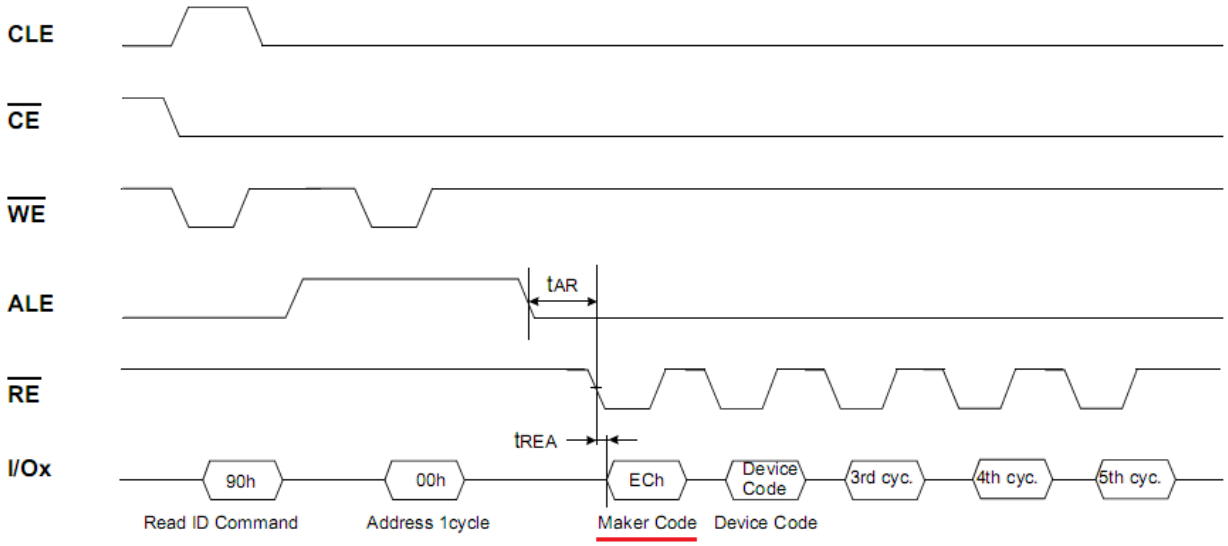
PG9=CE3 NAND
#define Bank1_LCD_C ((uint32_t)0x80000000)
#define FSMC_Bank3_NAND ((uint32_t)0x00000100)

플래시 메모리에는 셀을 연결한 논리 구조에 따라 여러 가지 종류가 있다. 대표적으로 셀을(플로팅 게이트를) 병렬 구조로 연결한 것이 NOR Flash Memory(노어 플래시 메모리)이고, 셀을 직렬 구조로 연결한 것이 바로 NAND Flash Memory(낸드 플래시 메모리) 이다.

1.Nand Memory

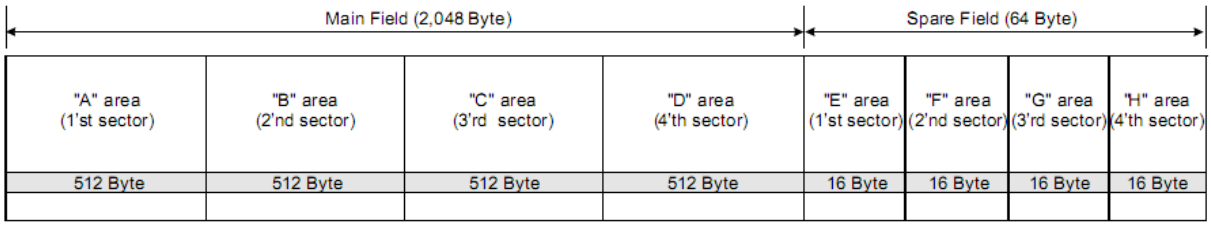
| 기능 | K9F2G08U0A 번호 | STM32F4 핀 이름 |
|---|---|---|
| DA0 | 29 | 85(PD14) |
| DA1 | 30 | 86(PD15) |
| DA2 | 31 | 114(PD0) |
| DA3 | 32 | 115(PD1) |
| DA4 | 41 | 58(PE7) |
| DA5 | 42 | 59(PE8) |
| DA6 | 43 | 60(PE9) |
| DA7 | 44 | 63(PE10) |
| WAIT | 7 | 122(PD6) |
| OE | 8 | 118(PD4) |
| WE | 18 | 119(PD5) |
| CLE (AD16) | 16 | 80(PD11) |
| ALE (AD17) | 17 | 81(PD12) |
| CE3 | 8 | 124(PG9) |

**Read ID Operation**

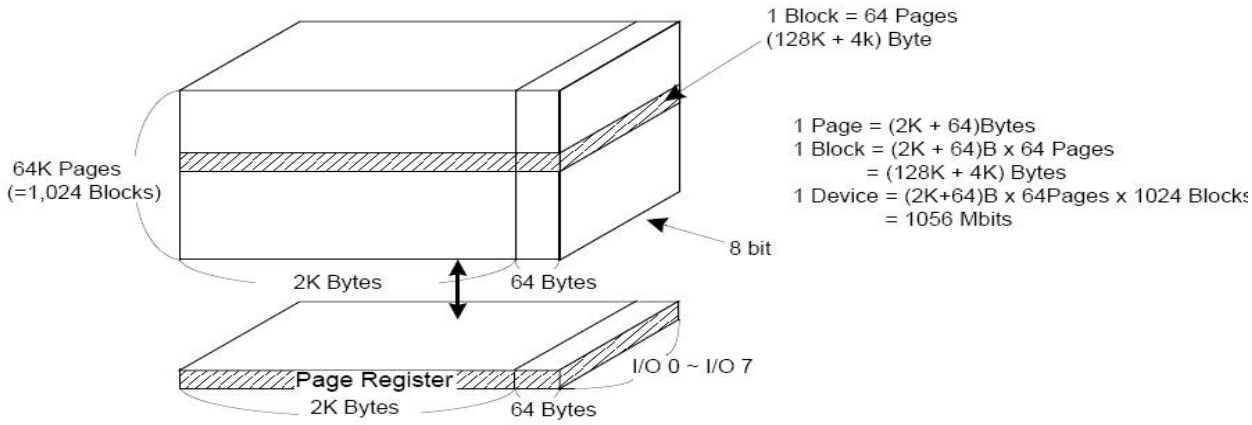



| Device | Device Code (2nd Cycle) | 3rd Cycle | 4th Cycle | 5th Cycle |
|---|---|---|---|---|
| K9F2G08R0A | AAh | 00h | 15h | 44h |
| K9F2G08U0A | DAh | 10h | 95h | 44h |

| | | |
|---|---|---|
| AHB3 | 0xA000 0000 - 0xA000 0FFF | FSMC control register |
| | 0x9000 0000 - 0x9FFF FFFF | FSMC bank 4 |
| | 0x8000 0000 - 0x8FFF FFFF | FSMC bank 3 |
| | 0x7000 0000 - 0x7FFF FFFF | FSMC bank 2 |
| | 0x6000 0000 - 0x6FFF FFFF | FSMC bank 1 |

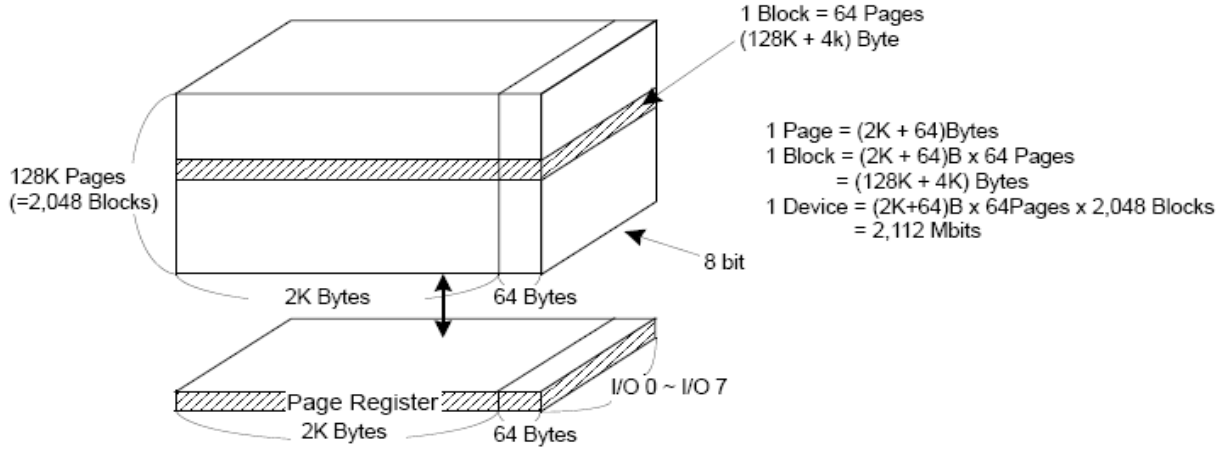


K9F1G08U




| | I/O 0 | I/O 1 | I/O 2 | I/O 3 | I/O 4 | I/O 5 | I/O 6 | I/O 7 | |
|---|---|---|---|---|---|---|---|---|---|
| 1st Cycle | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | Column Address |
| 2nd Cycle | A8 | A9 | A10 | A11 | *L | *L | *L | *L | Column Address |
| 3rd Cycle | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | Row Address |
| 4th Cycle | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | Row Address |

K9F2G08U




| | I/O 0 | I/O 1 | I/O 2 | I/O 3 | I/O 4 | I/O 5 | I/O 6 | I/O 7 | |
|---|---|---|---|---|---|---|---|---|---|
| 1st Cycle | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | Column Address |
| 2nd Cycle | A8 | A9 | A10 | A11 | *L | *L | *L | *L | Column Address |
| 3rd Cycle | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | Row Address |
| 4th Cycle | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | Row Address |
| 5th Cycle | A28 | *L | *L | *L | *L | *L | *L | *L | Row Address |

2. Nor Memory

| 핀번호 | ADDRESS | 핀번호 | DATA |
|---|---|---|---|
| PF0 | AD0 | PD14 | DA0 |
| PF1 | AD1 | PD15 | DA1 |
| PF2 | AD2 | PD0 | DA2 |
| PF3 | AD3 | PD1 | DA3 |
| PF4 | AD4 | PE7 | DA4 |
| PF5 | AD5 | PE8 | DA5 |
| PF12 | AD6 | PE9 | DA6 |
| PF13 | AD7 | PE10 | DA7 |
| PF14 | AD8 | PE11 | DA8 |
| PF15 | AD9 | PE12 | DA9 |
| PG0 | AD10 | PE13 | DA10 |
| PG1 | AD11 | PE14 | DA11 |
| PG2 | AD12 | PE15 | DA12 |
| PG3 | AD13 | PD8 | DA13 |
| PG4 | AD14 | PD9 | DA14 |
| PG5 | AD15 | PD10 | DA15 |
| PD11 | AD16 | | |
| PD12 | AD17 | PD4 | OE |
| PD13 | AD18 | PD5 | WE |
| PE3 | AD19 | PG10 | NE3 |
| PE4 | AD20 | PE0 | NBL0 |
| PE5 | AD21 | PE1 | NBL1 |
| PE6 | AD22 | | |

3.SRAM Memory

IS61WV102416BLL-10TLI  256M * 8Bit

| 핀번호 | ADDRESS | 핀번호 | DATA |
|---|---|---|---|
| PF0 | AD0 | PD14 | DA0 |
| PF1 | AD1 | PD15 | DA1 |
| PF2 | AD2 | PD0 | DA2 |
| PF3 | AD3 | PD1 | DA3 |
| PF4 | AD4 | PE7 | DA4 |
| PF5 | AD5 | PE8 | DA5 |
| PF12 | AD6 | PE9 | DA6 |
| PF13 | AD7 | PE10 | DA7 |
| PF14 | AD8 | PE11 | DA8 |
| PF15 | AD9 | PE12 | DA9 |
| PG0 | AD10 | PE13 | DA10 |
| PG1 | AD11 | PE14 | DA11 |
| PG2 | AD12 | PE15 | DA12 |
| PG3 | AD13 | PD8 | DA13 |
| PG4 | AD14 | PD9 | DA14 |
| PG5 | AD15 | PD10 | DA15 |
| PD11 | AD16 | PD4 | OE |
| PD12 | AD17 | PD5 | WE |
| PD13 | AD18 | PG12 | NE4 |
| PE3 | AD19 | PE0 | NBL0 |
| | | PE1 | NBL1 |