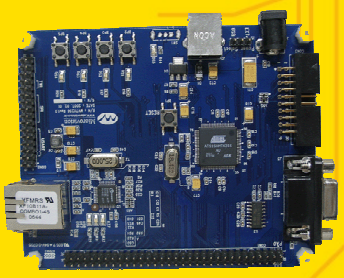
EWARM Educational Manual

Embedded On Demand

Microvision Co.,Ltd.

Embedded Total Solution / Emulator/ Compiler
Solution Board / Development Solution



㈜마이크로비젼 / Microvision Co.,Ltd.

서울특별시 구로구 구로3동 235번지 한신IT타워 610호

[☎] 02-3283-0101, [*] sale@microvision.co.kr

ATMEL SAM7X256 based

EWARM Educational Manual

MICROVISION Co.,Ltd.

Document Information

Version	0.75
File Name	IAR-EWARM_Educational_Manual(MV-7X256).doc
Date	2007.01.17.
Satus	Working

Revision History

Date	Version	Update Descriptions	Editor
2007.01.17.	V0.75	First Edition	MH.YOO
2007.5.2	V0.8	Second Edition	DH.Yoon

ATMEL SAM7X256 based EWARM Educational Manual

Published by MicroVision Co.,Ltd.

(☎) +82-2-3283-0101, (*) sale@microvision.co.kr

http://www.microvision.co.kr, http://www.mvtool.co.kr

Room #610, Hanshin IT Tower 235, Guro3-dong, Guro-gu, Seoul, Korea.

Contents.....

부록 **A.** 실습 예제 소스 정답

부록 **B. JTAG** 디버깅 환경 구축 – **C-SPY**

1. 제품(IAR EWARM) 개요

1.1 제품 개요

본 제품(제품명: **IAR EWARM**)은 세계적인 임베디드 소프트웨어 컴파일러 전문기업인 **IAR** 社 에서 개발한 **Embedded Workbench** 시리즈의 한 종류로서, **ARM7 Core** 기반의 **Smart ARM(SAM)** 시리즈를 개발한 **ATMEL** 社 홈페이지에서 **SAM** 프로세서 시리즈의 대표 컴파일러로 추천할만큼 코드의 압축율과 컴파일러의 안정성 및 편리성이 입증되었다.

IAR 社는 정책상 자사에서 판매되는 모든 컴파일러에 대해 사용자들이 테스트해 볼 수 있도록 데모 버전을 제공하고 있으며, **EWARM**의 경우 사용 기간 제한이 있는 **Evaluation Version** 과 사용 기간 제한은 없으나 **Code Size**에 제한이 있는 **KickStart Version** 으로 나뉘어져 있다.

본 교재에서는 내용의 범위가 넓은 만큼 사용 기간 제한이 없는 **KickStart Version** 을 기준으로 작성되어 있으므로 유의하도록 하자.

간략하게 **IAR EWARM**의 주요 특징**(feature)**들을 정리해 보면 다음과 같다.
참고로 아래 내용은 **IAR** 社 홈페이지**(www.iar.com)**에 있는 **Key Components** 내용을 발췌하였다.

- **Integrated development environment with project management tools and editor**
- **Highly optimizing ARM compiler supporting C and C++**
- **Configuration files for ARM chips from Analog Devices, Atmel, Cirrus Logic, Freescale, Intel, Luminary, NetSilicon, NXP, OKI, Samsung, Sharp, STMicroelectronics and Texas Instruments**
- **Extensive JTAG and RDI debugger support**
- **Optional IAR J-Link and IAR J-Trace hardware debug probes**
- **Run-time libraries including source code**
- **Relocating ARM assembler**
- **Linker and librarian tools**

- **C-SPY debugger with ARM simulator, JTAG support and support for RTOS-aware debugging on hardware**
- **Evaluation edition of IAR PowerPac RTOS and file system bundle**
- **RTOS plugins available from IAR Systems and RTOS vendors**
- **Code templates for commonly used code constructs**
- **Sample projects for evaluation boards from many different manufacturers**
- **User and reference guides, both printed and in PDF format**
- **Context-sensitive online help**

또한, 현재 버전에서 두드러지게 중요한 부분으로는 다음과 같다.

- **IAR PowerPac bundled evaluation edition of RTOS and file system for ARM**
- **Live watch on target hardware**
- **Code coverage using IAR J-Trace**
- **Comprehensive flash loader support**
- **I/O register definition files**
- **More than 400 sample projects for different evaluation boards**

þ 참고로 **IAR EW Compiler** 시리즈의 국내 총판은 ㈜마이크로비젼이 담당하고 있으며, 판매 및 납품 이외에 컴파일러에 관한 기술지원을 위하여 내부적으로 별도의 기술지원팀**(TST)**을 운영하고 있어 온라인**(E-mail)**이나 오프라인 상으로 기술지원을 책임지고 있다.

1.2 제품 구성(Package)

기존에 납품되던 **IAR EWARM**의 아이템별 구성 내역은 다음과 같다.

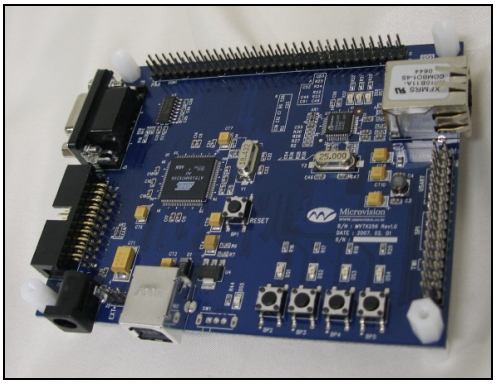
<표 **1-1> IAR EWARM** 제품 구성 내역

Item	Description
CD	구매한 소프트웨어 설치 패키지 **(PDF** 매뉴얼 포함**)**
Dongle	라이센스 동글키
Manual(영문**)**	**EWARM IDE User Guide, C/C++ Guide, Assembler Guide**
Guide	**Quick Start Guide(IAR), Install Guide, SUA** 안내문**(**마이크로비젼**)**
License	제품 인증서

이외에 본 교재인 한글판 매뉴얼도 이번에 새로 포함되었다.

2. MV7X256 소개

2.1 MV7X256 개요



본 교재에 수록된 **EWARM**의 모든 예제 소스들은 임베디드 솔루션 전문업체 ㈜마이크로비젼 **(www.microvision.co.kr)**에서 **ATMEL** 社의 **ARM7 Core** 기반인 **AT91SAM7X256** 프로세서를 탑재하여 **IAR** 社의 **EWARM** 컴파일러 교육용으로 개발한 **MV7X256 Kit**에서 모두 검증되었다.

ATMEL AT91SAM7X256 프로세서를 장착한 **MV7X256** 보드는 임베디드 컴파일러로 전세계적으로 널리 알려져 있는 **IAR** 社의 **EWARM** 컴파일러를 교육하기 위해 개발된 교육용 보드로 교육에 필요한 **I/O** 주변 장치, 시리얼**(UART)** 인터페이스와 같은 기본적인 주변장치들 이외에도 **TCP/IP(**인터넷**)** 연결을 **Firmware** 레벨에서 구현해 볼 수 있도록 **SAM7X256**에 내장된 **10/100Mbps** 이더넷**(Ethernet)** 인터페이스를 가지고 있다는 특징이 있다.

ARM9 Core가 아닌 **ARM7 Core**를 채택한 이유는 **ARM9** 이상의 레벨에서는 대부분 운영체제**(OS)**를 사용하게 되는데, 이때 사용되는 운영체제**(OS)**가 **Embedded Linux** 나 **Windows CE** 인 경우엔 **GCC, VC++** 와 같은 운영제체**(OS)** 전용 컴파일러를 사용하고 있기 때문에 **EWARM**을 사용하는 일이 거의 드물다.

따라서, 본 교재에서는 **EWARM**의 주요 고객들이 사용하고 있는 **ARM7 Core**를 기반으로 구성하고자 하였으며, 실습 예제를 풍부히 하고자 **TCP/IP(**인터넷**)** 연결까지 테스트해 볼 수 있는 **ATMEL** 社의 **AT91SAM7X256** 프로세서를 채택하여 보드를 구성하였다.

이러한 **MV7X256**의 특징**(feature)**들을 간략히 정리해 보면 다음과 같다.

- **10/100Mbps Ethernet MAC**이 내장된 **ATMEL AT91SAM7X256** 탑재
- **256KB Flash, 64KB SRAM** 내장
- **USB 2.0 Full Speed(12Mbps) Device** 포트 내장
- **10/100 base-T Ethernet MAC** 내장

– **AT91SAM7X256** 프로세서 핀들의 핀들에 대한 **1:1** 확장 커넥터 지원

4. 개발 환경 구축

4.1 MV7X256 소프트웨어 개발 환경 구축

4.1.1 EWARM 이란?

EWARM 이란, **IAR** 사에서 개발된 **IAR Embedded Workbench** 시리즈의 한 종류이다. 컴파일러 개발 전문 회사 **IAR** 社**(www.iar.com)**는 현재 **8/ 16/ 32 bit Microprocessor** 와 **DSP** 등 **30**가지 이상의 **Compiler** 시리즈를 전세계를 대상으로 지원**/**판매하며**,** 안정적인 **Tool** 과 우수한 컴파일러 환경을 제공하고 있다**.**

EWARM compiler 는 **ARM7/9/9E/10/11** 시리즈 계열을 모두 지원하며**, C-Spy (IAR S/W Debugger** 환경**)**와 **H/W Emulator**를 연동한다면 쉽고**,** 빠른 **Debugging** 환경을 제공 받을 수 있다**.** 뿐만 아니라**,** 칩 社 별로 제공되는 각종 **Evaluation boards** 와 **Example Project Source** 는 **EWARM Compiler**를 바로 활용하여 기본적인 주변 **Peripheral**을 제어해 볼 수 있는 사용자 편의를 제공하고 있다**. IAR Complier** 데모 버젼은 **EWARM 4.41A** 관련 **2**가지 제공되며 **Evaluation Version**과 **KickStart Version** 의 사양은 아래와 같다**.**

<표 **4-1> IAR EWARM 4.41A Demo Version** 사양 **(* 2007**년 **2**월 현재**)**

제한 사항	Evaluation Version	KickStart Version
사용 기간	**30 days**	무제한 **(**약 **25**년 제한**)**
Code Size	**No Limit**	**32KB(byte) Limit**
기타	**– Runtime Libraries Source Code** 미 포함 **– MISRA C** 미 포함 **– Technical Support Limit by IAR**	

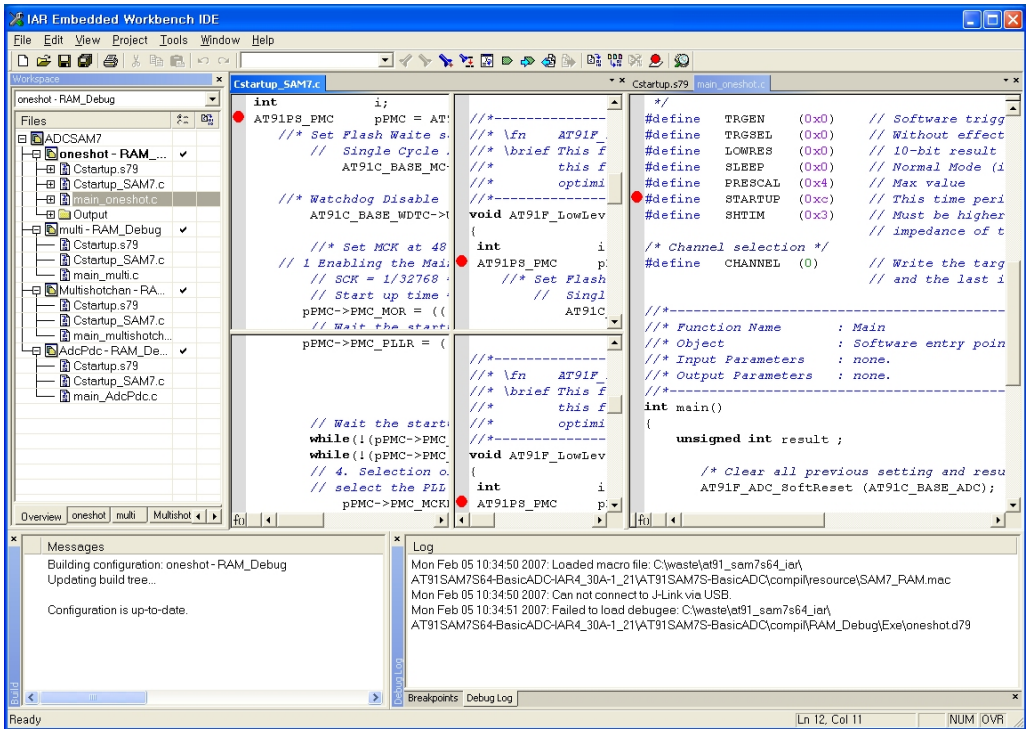
본 교재에서는 **Reference** 교재 특성상 기한 제한을 두지 않은 **KickStart Version** 으로 설치하도록 함으로써**,** 사용 중 갑자기 **Compile**이 되지 않는 불가피한 상황을 피하도록 하겠다**.** 물론**, 1**달 내에 모든 습득이 가능하거나**, Code Limited**에 대한 제한사항이 부담이 된다면 **Evaluation Version** 을 설치해도 설치 및 설정내용은 큰 차이가 없도록 구성하였다**.**

※ **[**주의**] Evaluation** 버전의 경우**,** 설치된 **PC**에선 재 설치가 되지 않으므로 주의해야 한다**.**

R EWARM : ARM 시리즈용 **Cross Compiler/Debugger**

R 공식 홈페이지 **: IAR** 본사**(www.iar.com)**

R IAR Korea Sole Agent : ㈜마이크로비젼**(www.microvision.co.kr)**



<그림 **3.1> EWARM** 실행 화면

3.2.2 EWARM Demo Download

1) IAR 공식 홈페이지**(www.iar.com)** 혹은**, IAR** 공식 대리점 마이크로비젼 **(www.microvision.co.kr)**을 통해서 다운 받을 수 있다**.**

IAR 공식 홈페이지 **Download** 경로 **:**

- Downloads > Evaluation Software >

IAR Embedded Workbench – 30 day evaluation edition > ARM

- Downloads > Evaluation Software >

IAR Embedded Workbench – KickStart edition > ARM (32KB edition)마이크로비젼 홈페이지 **Download** 경로 **:**

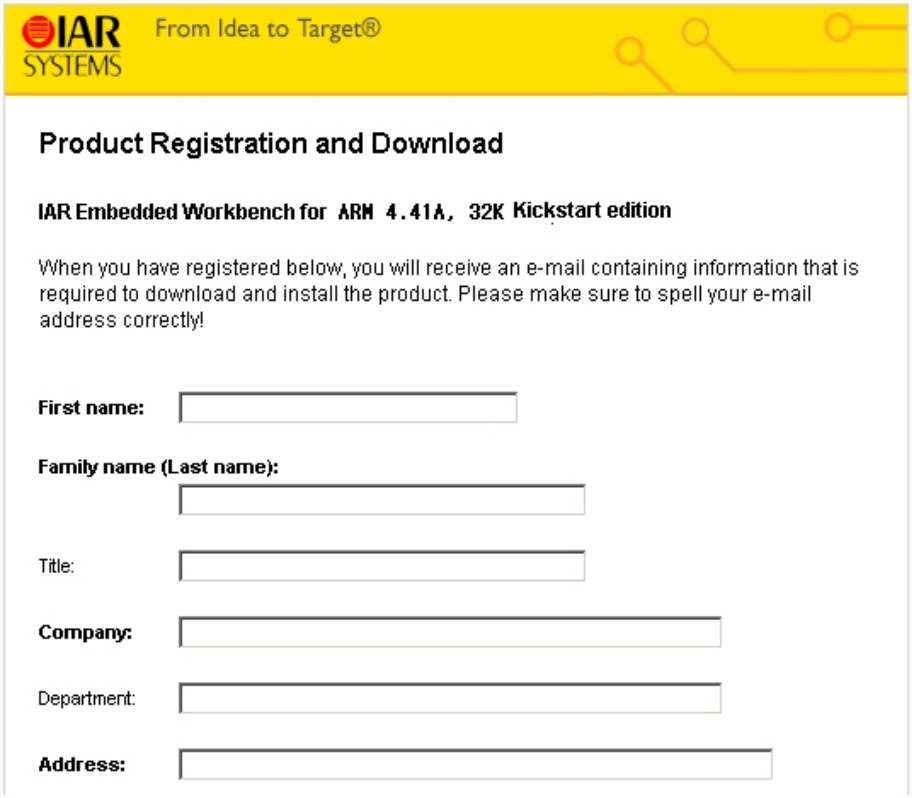
- 제품소개 **Products > 컴파일러/디버거 > EW-Compiler >**

(목록 중**) 컴파일러 종류 > ARM architecture – ARM** 클릭

(IAR Home page LINK)

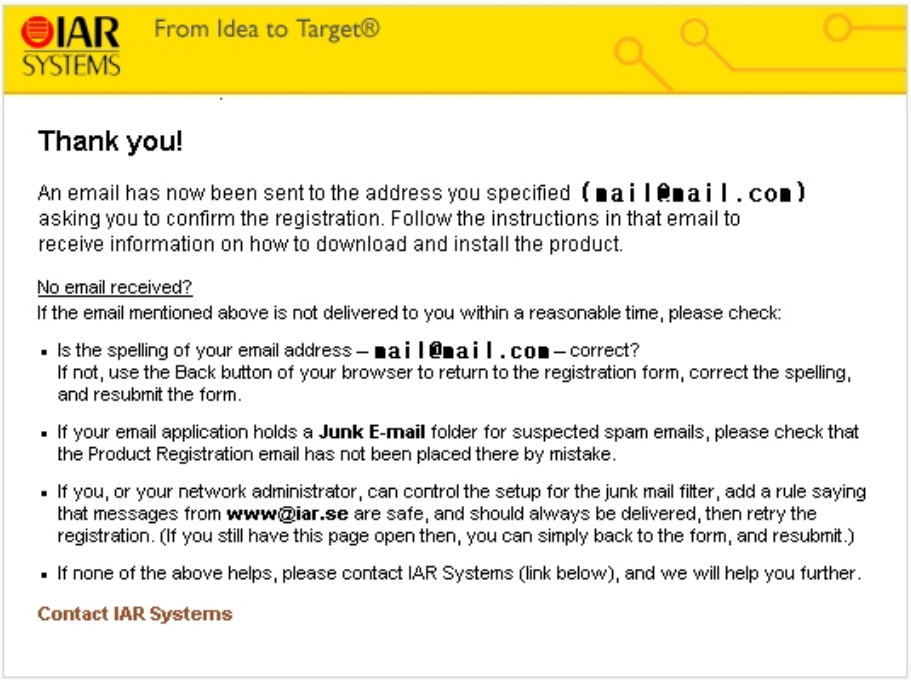
2) EWARM 4.41A KickStart Demo Version 설치 시에는 반드시 **License** 와 **License Key** 를 입력을 해야만 한다**. License**는 **IAR** 홈페이지에서 **Demo** 버전을 다운 받을 경우 무상으로 발급된다**.**

3) IAR의 모든 **Demo** 버전은 **License**를 발급 받기 위한 **Product Registration (**사용자 등록과정**)**을 거쳐야 한다**.** 따라서**, License** 가 발급되는 **E-mail** 주소는 정확하게 작성되어야만 한다**.**



<그림 3.2> EWARM Demo 사용자 등록 과정

4) 모든 등록과정이 마쳤다면**,** 가장 하단의 **[Submit Registration]**을 누른다**. Demo** 사용에 대한 감사 메시지와 함께 사용자가 등록한 메일 주소가 정확한지 다시 한번 확인한다**.**



<그림 3.3 > 등록 확인 메시지

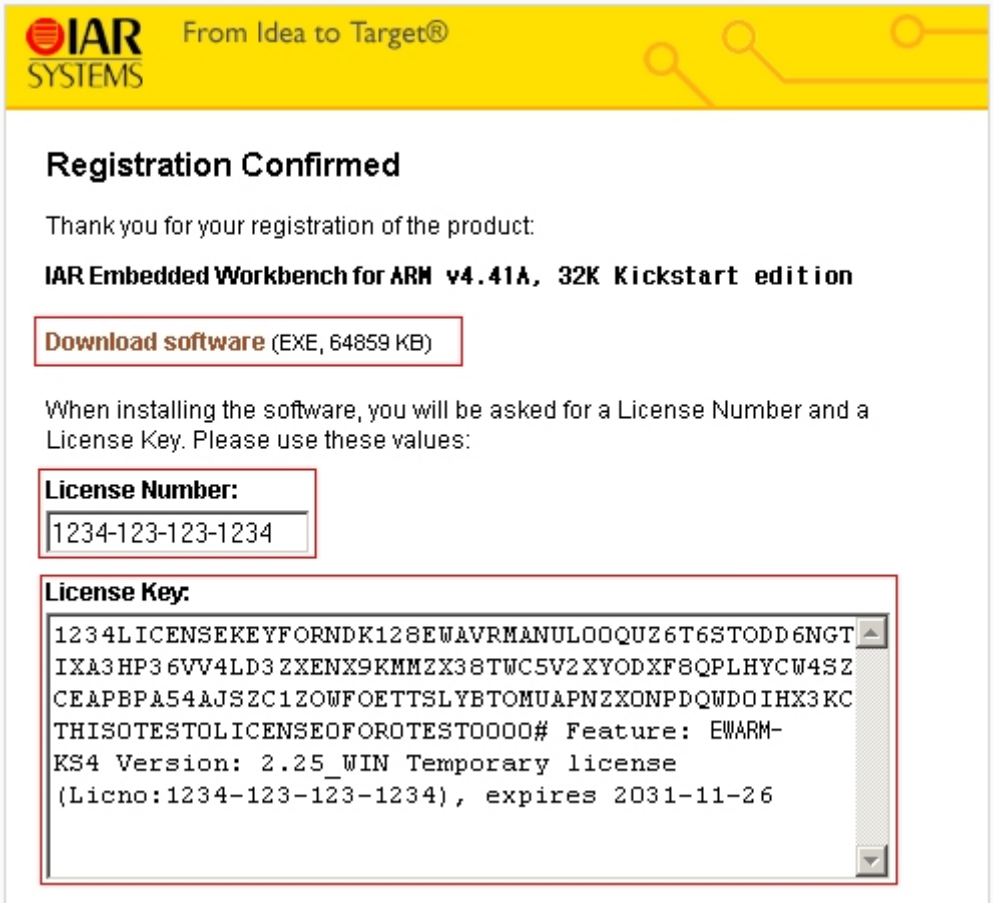
5) 확인 결과 문제가 없다면**,** 등록된 **E-mail**로 메일 수신이 되었는지 확인한다**.**
(<그림 **3.6>** 에서는 **mail@mail.com** 로 메일 등록된 예문 **)**

수신이 되었다면**,** 아래의 제목과 함께 메일이 수신되었음을 확인 할 수 있을 것이며**,**

"**Product registration: IAR Embedded Workbench for ARM v4.xx, 32K Kickstart edition** "

메일 수신 내용 중 링크가 되어있는 아래의 주소로 이동하도록 한다**.**
" http://supp.iar.com/Register/Confirm/?Reg=12341234…. " <<**클릭**

6) 이동한 **Page**에는 아래와 같이 다운로드 받을 수 있는 **Demo Software** 와 **License Number/ Key** 가 제공된다**.**



<그림 3.4 > 등록 확인 메시지

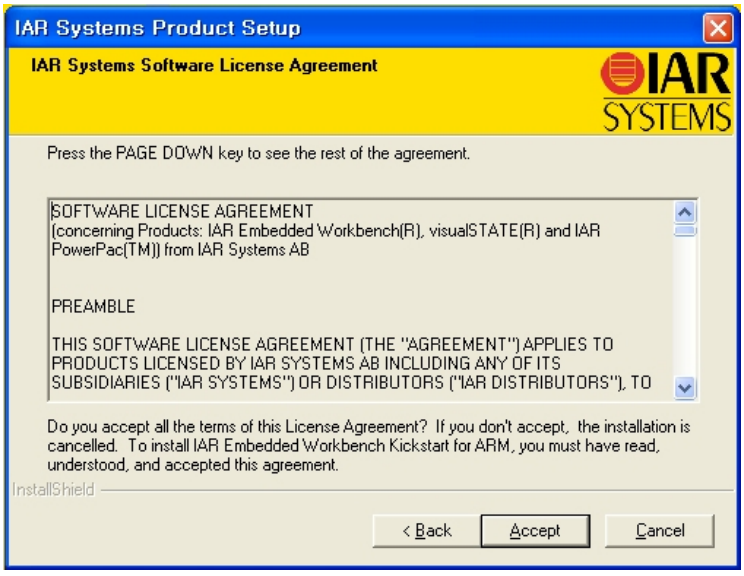
*** 이상의 내용은 EWARM Evaluation Version 에서도 동일하게 진행된다.**

3.2.3 EWARM 설치

1) 다운 받은 **EWARM-ks-web-441a.exe** 파일을 실행하면**,** 아래와 같이 설치 초기 화면이 나오며 **Next** 버튼을 누른다**.**

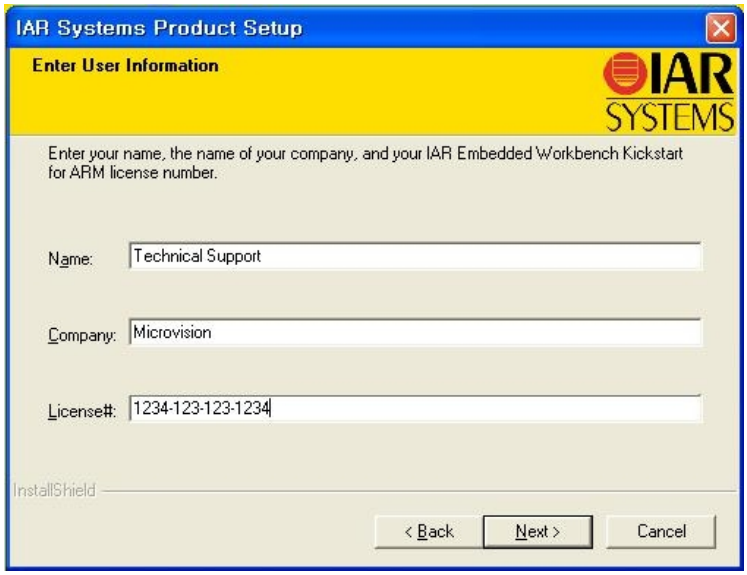
<그림 3.5> EWARM 초기 설치 화면

2) 라이센스에 동의한다면**, [Accrpt]** 버튼을 클릭한다**.**



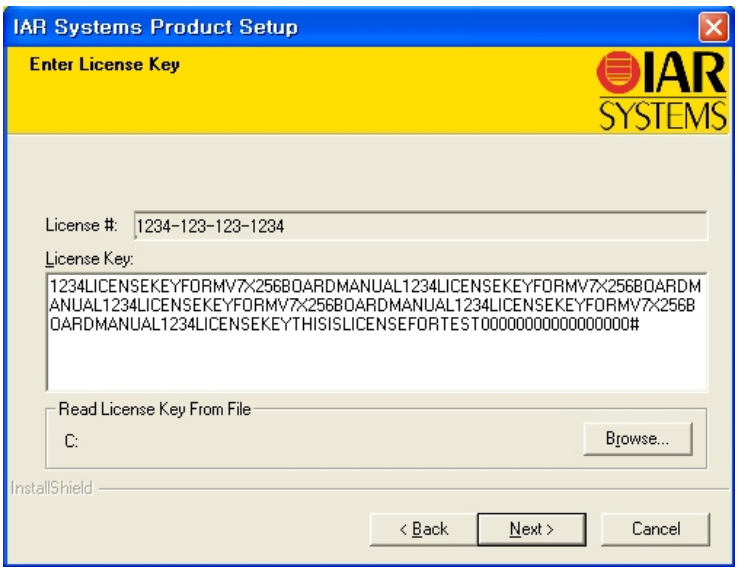
<그림 3.6 > EWARM 라이센스 동의 화면

3) 사용자 등록 과정으로써**,** 사용자 이름**/** 회사를 입력하고**,** 발급된 **License Number** 를 넣고서 **[Next]** 버튼을 클릭한다**.**



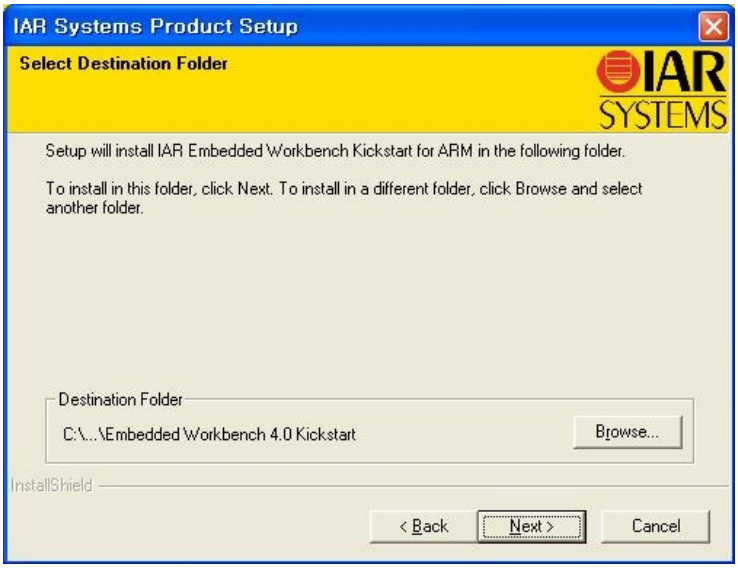
<그림 3.7 > 사용자 정보 및 License 등록 화면

4) License Number 와 함께 제공된 **License Key** 를 '붙여넣기' 하고 **[Next]** 버튼을 누른다**.**



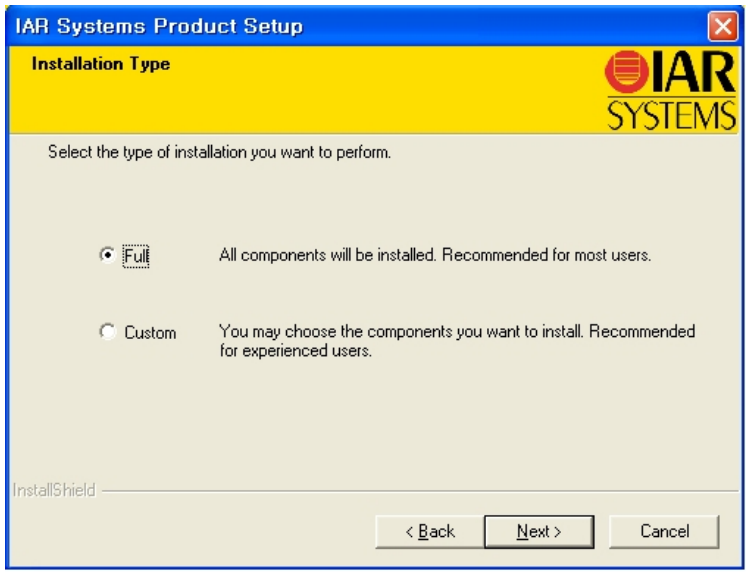
<그림 3.8> License Key 입력 화면

5) 설치 디렉토리를 지정한 뒤**, [Next]** 버튼을 누른다**.**



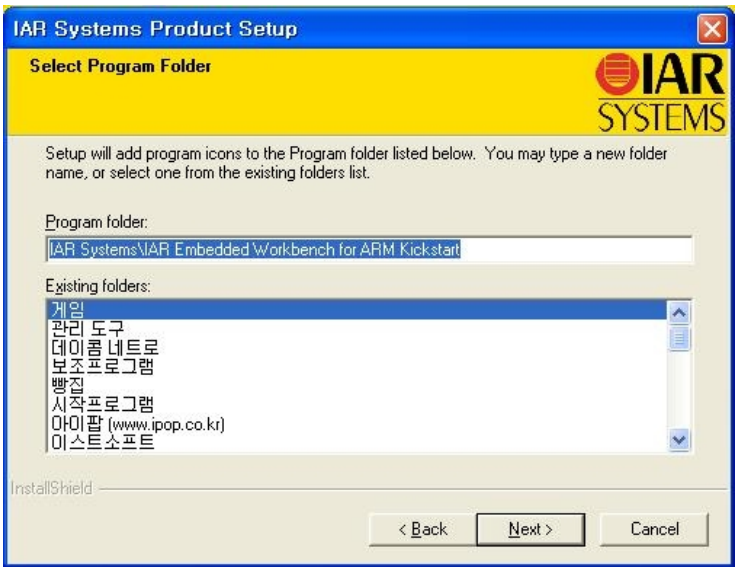
<그림 3.9> EWARM 설치 폴더 화면

6) C-Spy 등 부가 프로그램을 제한적으로 설치할 수 있으나**,** 교육 목적상 **[Full]** 선택 후 **Next** 버튼을 누른다**.**



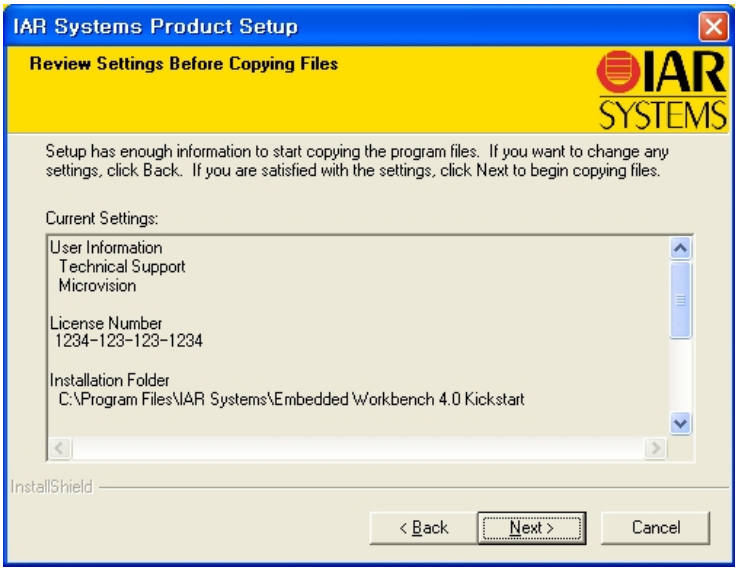
<그림 3.10> ARM 설치 타입 선택 화면

7) 설치 디렉토리는 확인 후**, [Next]** 버튼을 누른다**.**



<그림 3.11> EWARM 프로그램 폴더 선택 화면

8) 설정에 관한 내용을 다시 한번 확인하고**, [Next]** 버튼을 누르면**,** 프로그램 인스톨이 시작된다**.**



<그림 3.12> 설정 내용 확인 화면

9) 전 단계가 문제가 없다면 설치가 진행된다.

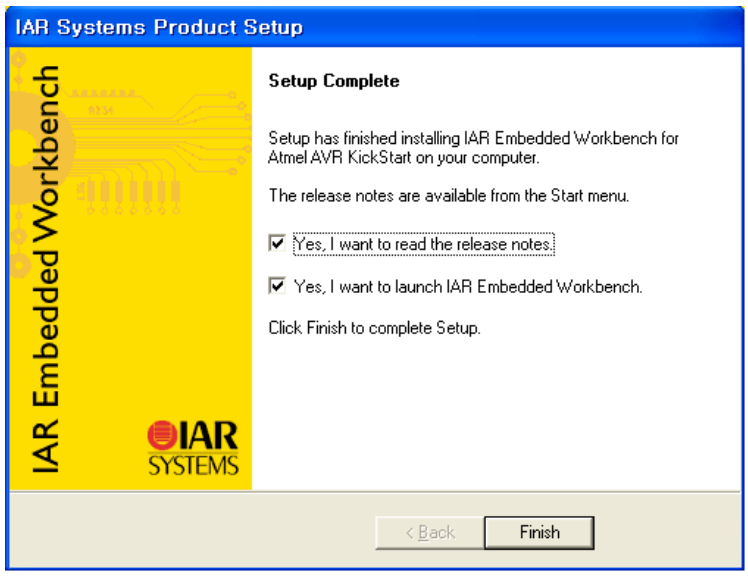
설치가 완료되면, 아래와 같음 화면을 볼 수 있다. 마지막으로 **Finish** 버튼을 클릭하면 설치가 완료된다.

- **Yes, I want to read the release notes.**

  각각의 컴파일러 버전에 관련된 **Upgrade** 정보를 보여준다.

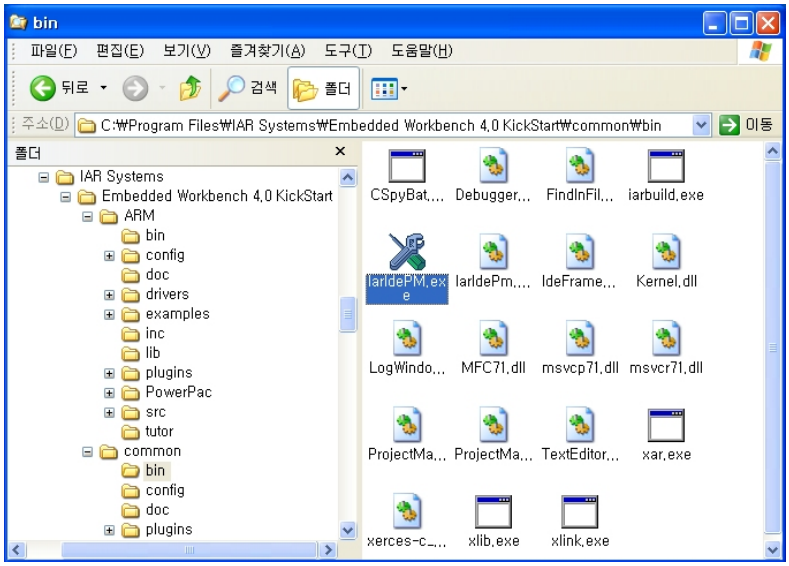
- **Yes, I want to launch IAR Embedded Workbench.**

  인스톨 프로그램을 종료하면서, 바로 컴파일러 프로그램 실행해준다.

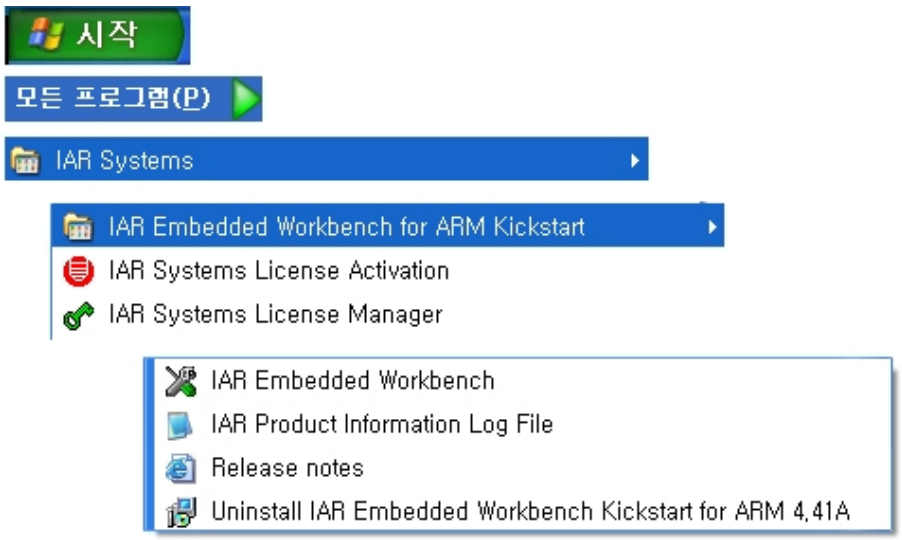


<그림 3.13> EWARM 데모 버젼 설치 완료 화면

10) [Finish] 버튼을 누르면, **Release Note** 자료와 **EWARM** 컴파일러가 실행된다.



<그림 3.14> EWARM 기본 설치 디렉토리 화면



<그림 3.15> 시작 메뉴에 등록된 EWARM 실행 화면

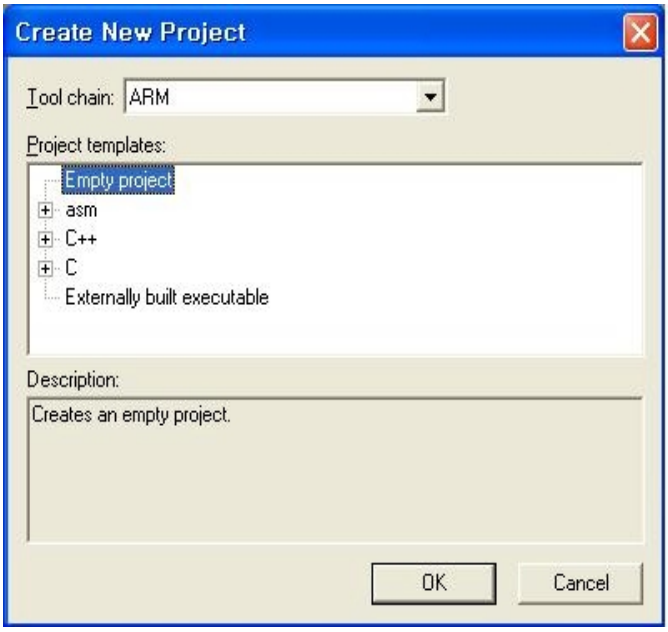
3.2.4 EWARM Compiler 설정

EWARM Compiler가 정상적으로 설치되어 문제없이 실행되는 것을 확인 했다면, **Target Board MV7x256 (Atmel AT91SAM7x256)** 맞춰서 사용하기 위해 다음과 같이 컴파일러의 환경 설정하도록 한다.

> ※ **[**참고**] IAR** 컴파일러는 기존 컴파일러의 프로젝트 개념에 **WorkSpace** 라는 프로젝트 관리자를 두고있다.
>
> **WorkSpace > Project > Source**
>
> **(*.eww) (*.ewp) (.c ; *.s* ; *.h ..)**

1) Project (*.ewp) 만들기

인스톨 후, 화면 중앙에 띄어져 있는 **[Embedded Workbench Startup]** 목록 중에 **[Create new Project in current workspace]** 를 선택한다.

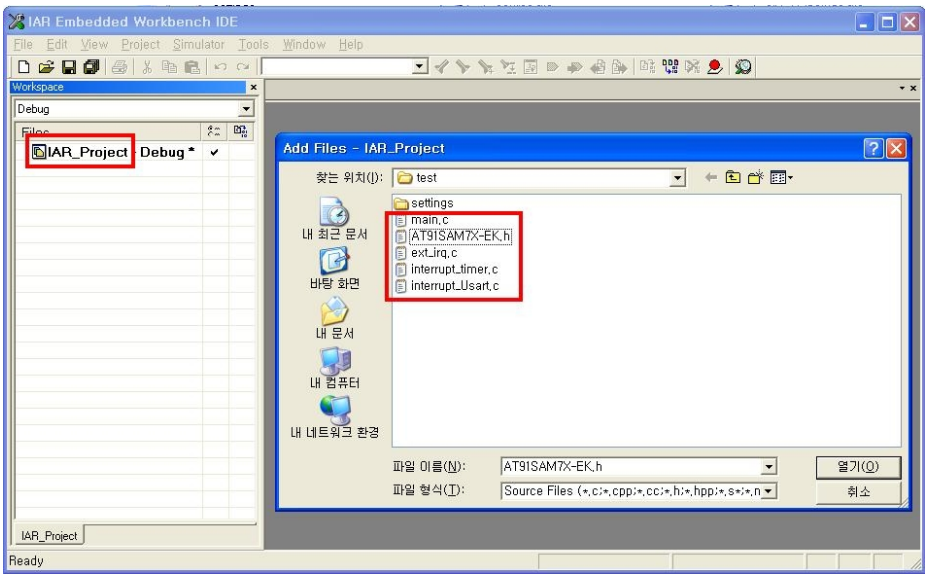


<그림 3.16> New Project Menu

Create New Project 에서 **Tool Chain** 이 **[ARM]**으로 선택하고,
[Empty Project]를 선택하여 '새로운 프로젝트'를 구성하도록 하자.

2) 생성된 **Project** 에 **Source files (.c ; *.s* ; *.h ..)** 추가하기

임의적으로 프로젝트 파일이 저장될 폴더와 프로젝트 명**(IAR_Project)**을 설정하면, 그림과 같이 화면 좌측의 **Workspace** 창 내에 프로젝트가 추가된다.

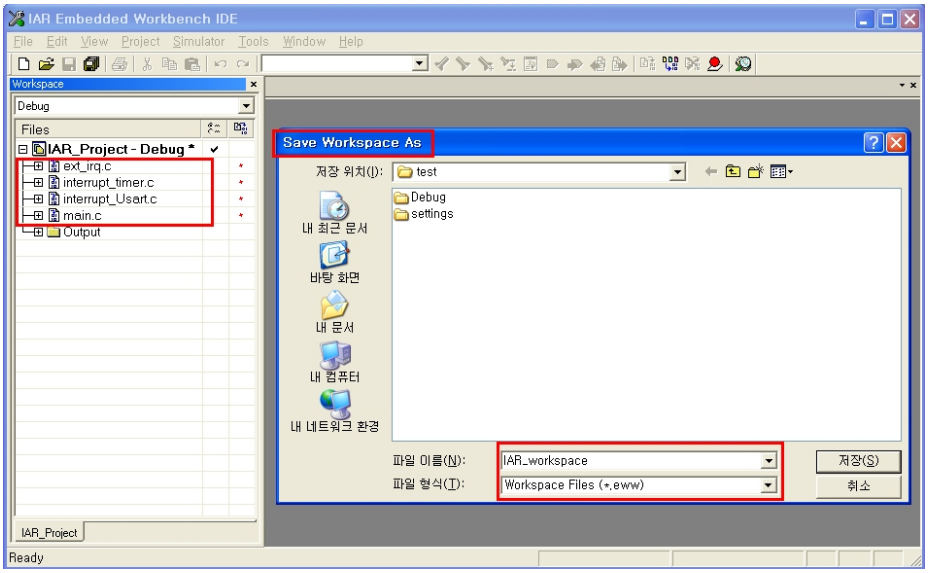


<그림 3.17> Add Source Files 화면

이렇게, 만들어진 프로젝트에 메뉴 **[Project] – [Add Files]**를 실행하면 제공된 소스를 추가할 수 있다. 마찬가지로, 좌측 창에 추가**(Add)**된 **Source**를 확인해 볼 수 있을 것이다.

3) Workspace file (*.eww)저장하기

마지막으로, 기본 **Project** 생성이 끝났다면, **[File] – [Save WorkSpace]**로 저장하도록 한다. **Workspace** 저장을 해야 컴파일을 진행할 수 있다.



<그림 3.18> WorkSpace 저장 화면

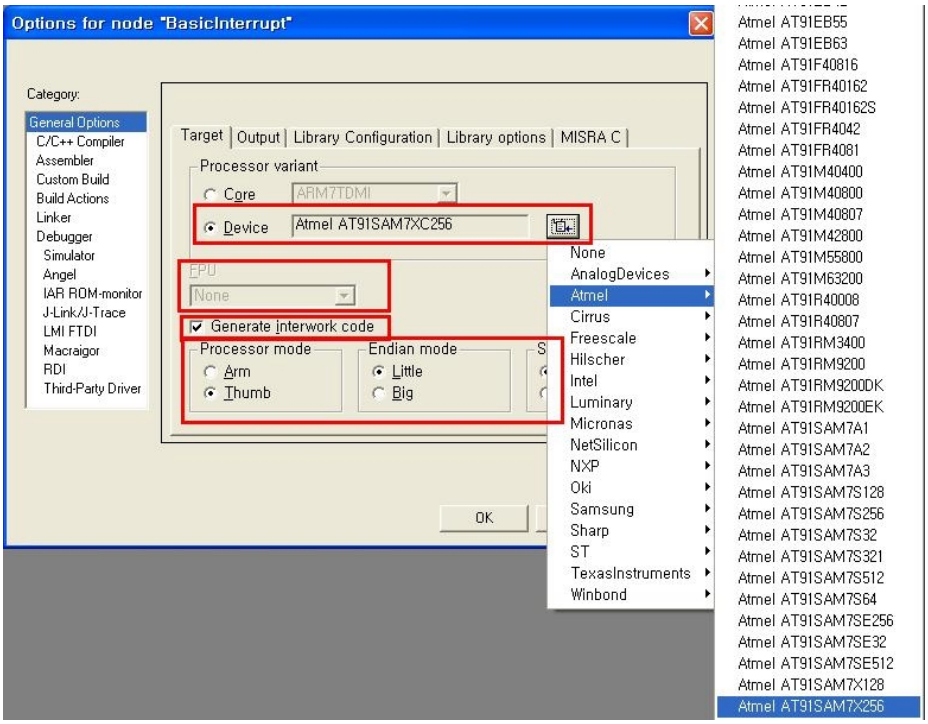
4) MV7X256 을 위한 **Project Option** 설정

기본적인 **Project** 구성을 마쳤다면**, H/W**와 **S/W** 구성에 맞게 **IAR Compiler Option** 설정이 필요하다**.**

> ※ **[**참고**]** MV7X256 은 외부 External Memory 를 사용하지 않고, 일반적인 Thumb 모드 지원사양으로 구성할 것이며, 디버깅 용 혹은, bin 이미지를 만들어 내기 위한 용도… "라는 등의 구상 하에 옵션을 설정하게 된다.

먼저**,** 가장 기본적인 **Option** 에 대해서 설명하도록 하겠다**.**

4-1) CPU 선정



<그림 3.19 > [Project] – [Options] – [General Options] – [Target] 화면

- **Processor Variant**

사용하게 될 **CPU** 를 선정하는 가장 중요한 옵션이다**.**

[Core] 목록에서 사용하는 **Device**의 **ARM** 계열을 선택하는 방법과**, [Device]** 명으로 선택하는 방법이 있다**.**

가급적 **Device** 명으로 간단하게 설정을 하는 방법을 권장하며**,** 만일**, Core** 로 선택할 경

우, 세부적인 옵션들을 일일이 확인해 볼 필요가 있다.

- **FPU [Disable]**

Float-Pointing unit 용으로써, **VFP** 코프로세서가 지원이 될 경우에만 사용할 수 있다.

(* VFP : Vector Floating pointing)

- **Generate interwork Code [Check]**

ARM mode 와 **Thumb Mode** 를 병행해서 사용할 경우 설정한다.

MV7x256 보드의 경우에는 반드시 체크하고 넘어가도록 하자.

- **Processor Mode/ Endian Mode / Stack Align**

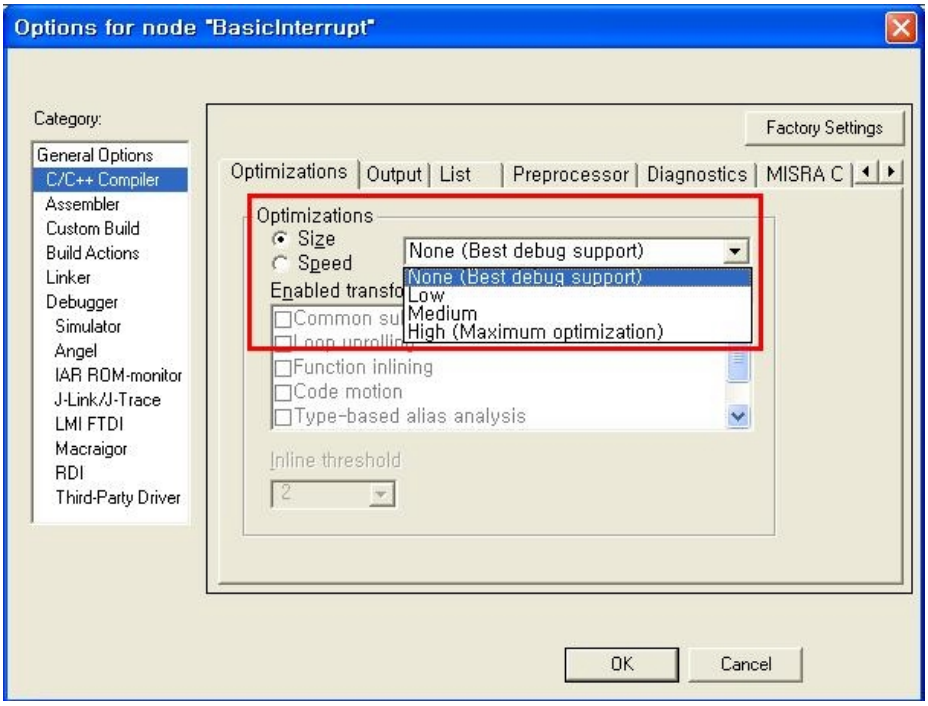
실제 사용하게 될 환경이 선택하도록 한다.

일반적으로 **SAM7x256 MCU**의 실제 프로세싱은 메모리 사이즈를 고려하여 **[Thumb]** 로 선택하고, **[Little Endian]** 방식과 **[4byte]** 을 선택한다.

> ※ **[**참고**]** ATMEL SAM7x256 MCU 는 실제 동작은 부족한 내부 Flash 메모리 사이즈를 고려하여, Thumb 모드로 실제 동작된다. 하지만, C-startup 전의 Interrupt Vector 선언부나 Handler 는 ARM mode (Instruction) 을 사용해야만 한다.

4-2) Optimization Level 설정

EWARM 의 **Optimization** 설정은 기본적으로 **[Size]** 와 **[Speed]** 로 구분되며, 각각의 **4** 가지의 **Level** 설정을 갖는다.



<그림 3.20> [Project] – [Options] – [C/C++ Compiler] – [Optimization] 화면

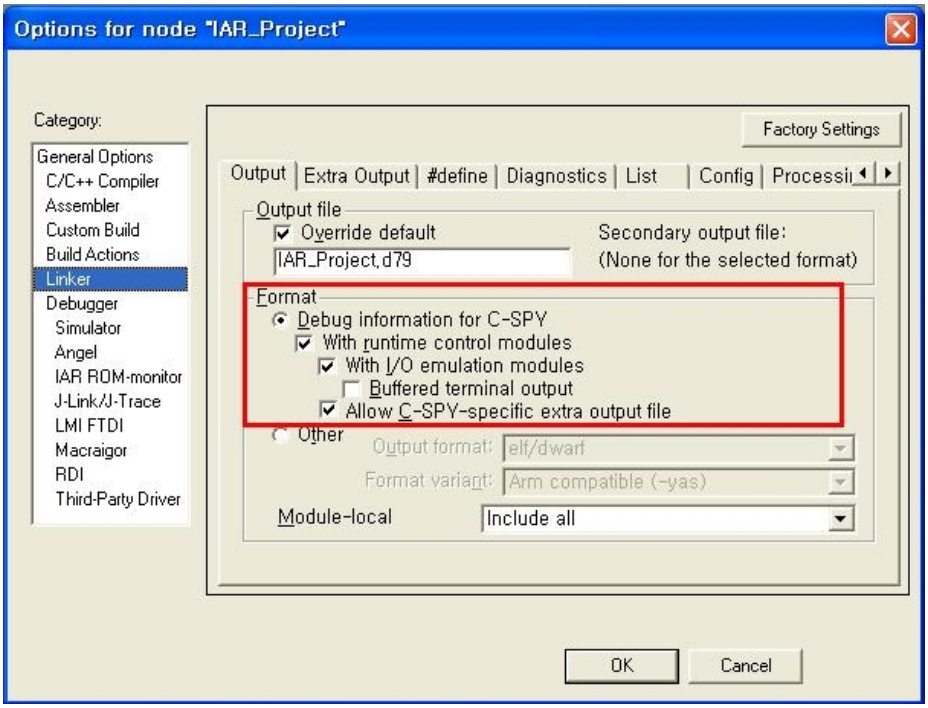
각각의 **Level** 은 괄호의 참고에도 명시되어 있다시피**, Format** 을 어떻게 할 것이냐에 따라서 구분하도록 한다**.**

- **None [Best Debug Support]**
- **Low**
- **Medium**
- **High [Maximum Optimization]**

***Debugging** 시에는 **[None]**으로 사용할 수 있도록 권장하며**, Debugging** 이 끝난 시점에서는 **Medium** 나 **High**로 설정하도록 한다**.**

4-3) Output 파일 선정

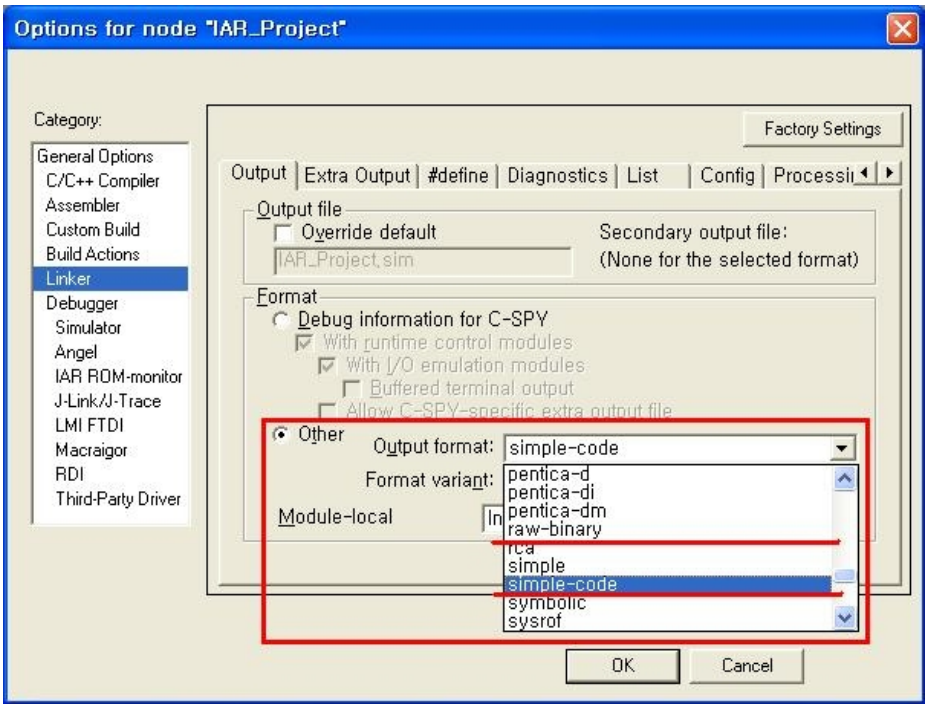
IAR Compiler는 기본적인 **C-Spy** 용 디버거 **Output** 되도록 설정되어 있으며**,** 이 디버거 용 파일**(*.d79)** 은 사용된 소스의 경로와 각종 **Symbol** 들의 정보가 포함이 되도록 되어있습니다**.**



<그림 3.21 > [Project] – [Options] – [Linker] – [Output] – C-Spy 화면

또한, 부가적인 **Output** 파일들을 별도로 제공하고 있습니다. 기타 여러 가지 **Format** 이 존재하지만, 기본적으로 사용되는 **Other Format** 은 **2**가지 정도로 요약할 수 있다.

- **Sample-code Format : Flash loader** 에 의해서 다운로드 되는 **binary file (32bit , Header file/ Starting point/ address information** 포함 **)**
- **Raw-binary Format : Flash Writing** 프로그램을 이용해서 다운로드 되는 일반적인 **pure binary file (32bit , Header file/ Starting point/ address information** 미포함 **)**



<그림 3.22 > [Project] – [Options] – [Linker] – [Output] - Other화면

- **Output file**

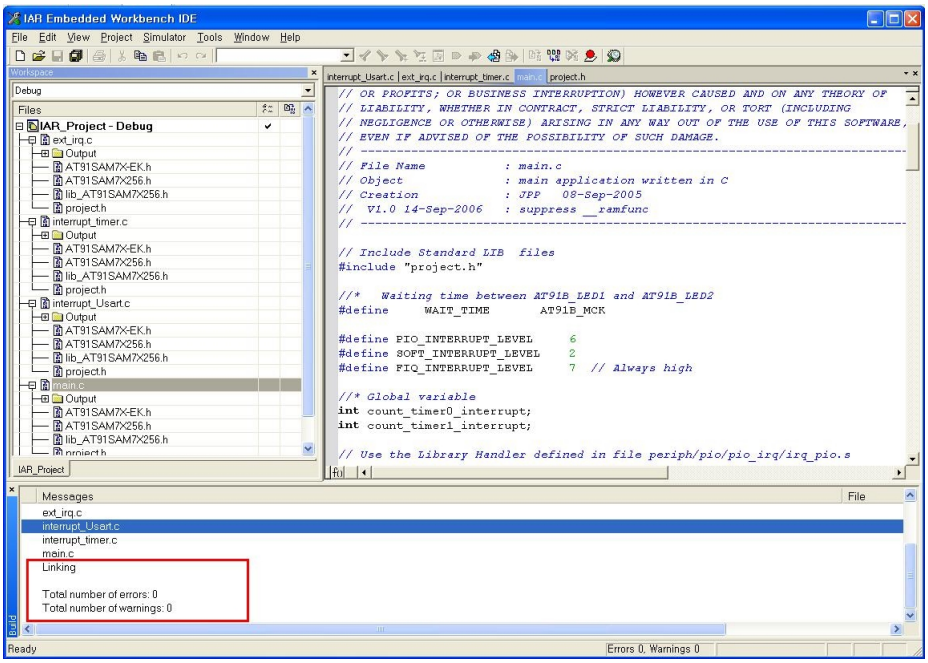
  컴파일 후 생성되는 실행파일의 파일명과 확장명을 지정할 수 있다.

- **Format**

  C-Spy 를 사용하기 위한 기본 옵션 설정들과 **Flash debugging** 을 하기 위한 **[Allow C-Spy specific extra output file]** 옵션이 있다.

> ※ **[참고] Flash Debugging** 시에는 반드시 **C-Spy Debugging** 옵션과 함께 **extra output** 파일로, **Simple-code** 옵션이 병행되어야만 합니다.

이로써, 기본적인 설정은 끝났으며, 메뉴 중 **[Project]–[Rebuild All]** 을 실행시켜 **Compile error** 유무를 체크 한다.



<그림 3.23> 에러 없이 성공적으로 컴파일이 완료된 화면

5. 실습 예제 #1

- Parallel Input/Output Controller(PIO)

5.1 SAM7X256 : Parallel Input/Output Controller(PIO)

ATMEL 7x256 의 PIO는 프로그래밍이 가능한 32개의 I/O line 이 존재하는데, [그림27-3]에서 보듯이 대부분의 control logic 들이 각각의 I/O 와 연결되어 동작되도록 되어있다.

Figure 27-3. I/O Line Control Logic

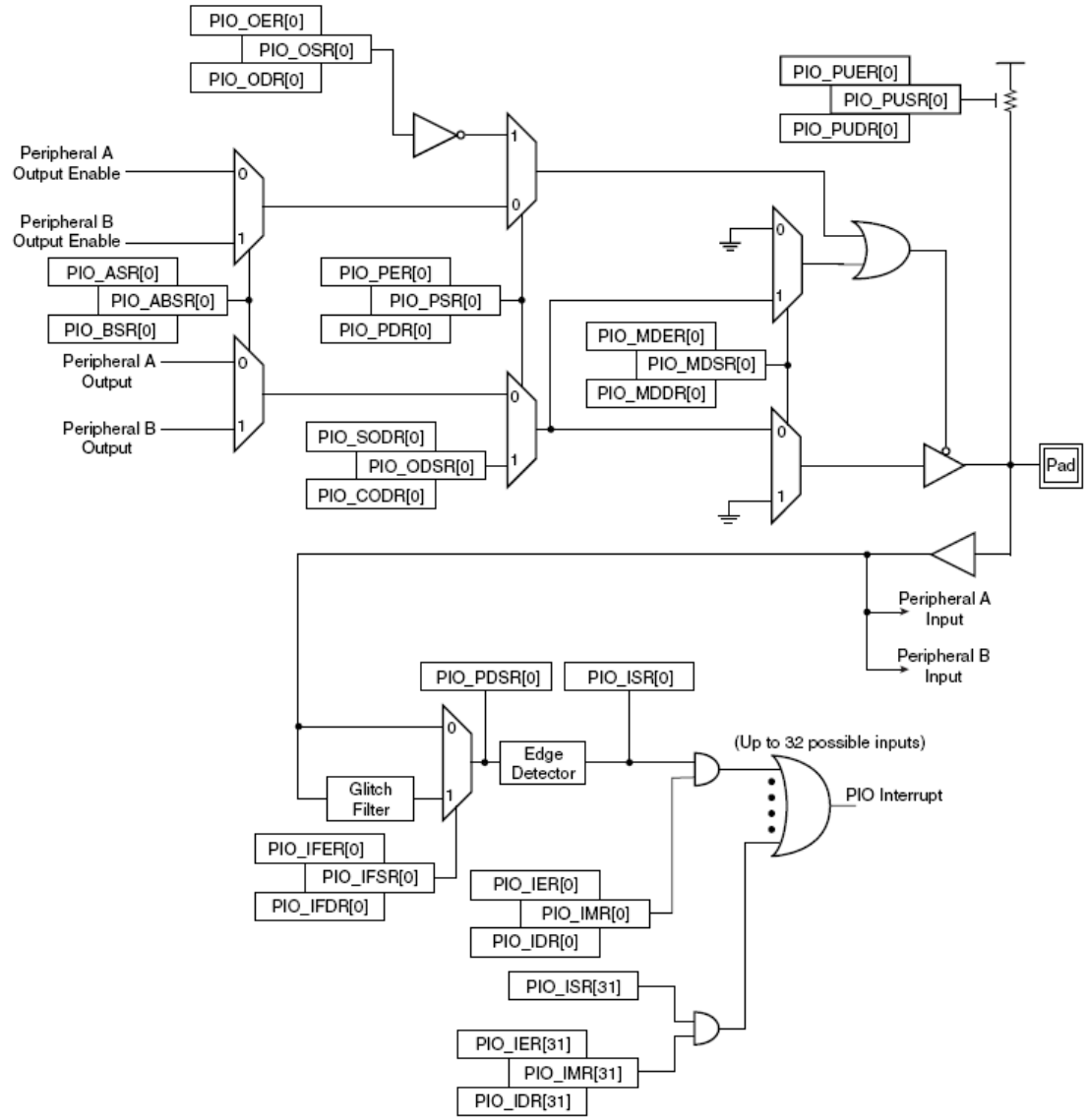


Table 27-2. Parallel Input/Output Controller (PIO) Register Mapping

Offset	Register	Name	Access	Reset Value
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register (1)	PIO_PSR	Read-only	0x0000 0000
0x000C	Reserved			
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	–
0x0038	Output Data Status Register(2)	PIO_ODSR	Read-only	0x0000 0000
0x003C	Pin Data Status Register(3)	PIO_PDSR	Read-only	
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register(4)	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved			
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	0x00000000
0x006C	Reserved			
0x0070	Peripheral A Select Register(5)	PIO_ASR	Write-only	–
0x0074	Peripheral B Select Register(5)	PIO_BSR	Write-only	–
0x0078	AB Status Register(5)	PIO_ABSR	Read-only	0x00000000
0x007C - 0x009C	Reserved			
0x00A0	Output Write Enable	PIO_OWER	Write-only	–
0x00A4	Output Write Disable	PIO_OWDR	Write-only	–
0x00A8	Output Write Status Register	PIO_OWSR	Read-only	0x00000000
0x00AC - 0x00FC	Reserved			

사용되는 모든 레지스터들은 lib_AT91SAM7X256.h 에 각각의 함수로 정의 되어 있으므로 간편하게 사용할 수가 있다.

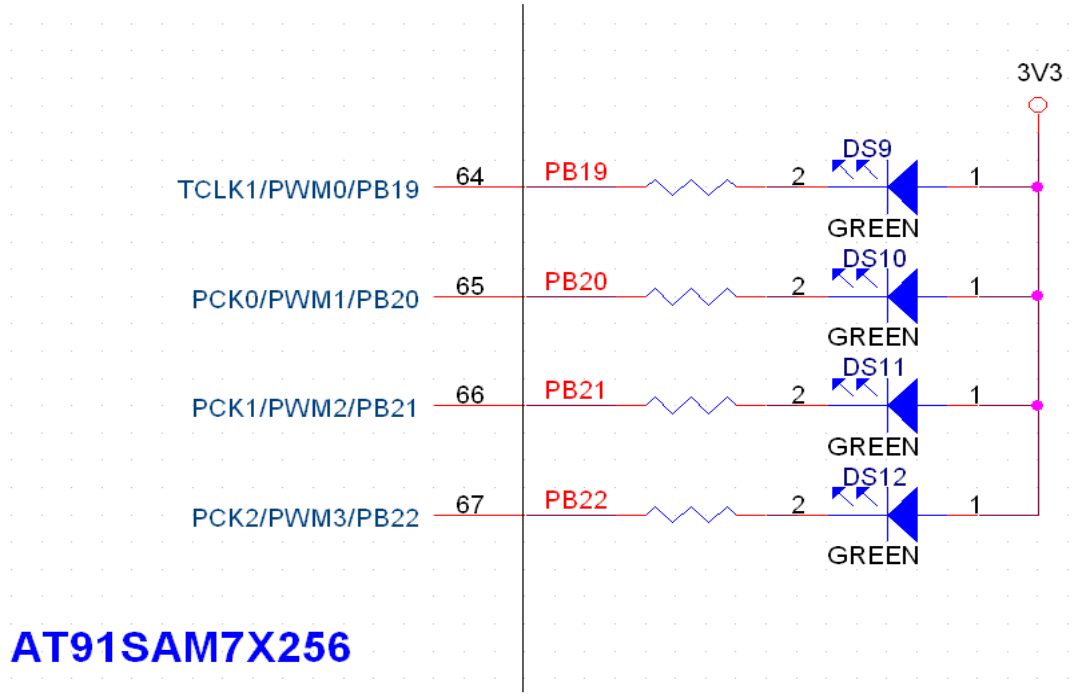
5.2 MV-7X256 PIO 회로 구성

MV-7X256 에서 PIO 테스트 용으로 사용된 입력 line은 PORTA 21/22/23/30 번, 출력 line 은

PORTB 19/20/21/22 번을 각각 사용했다.

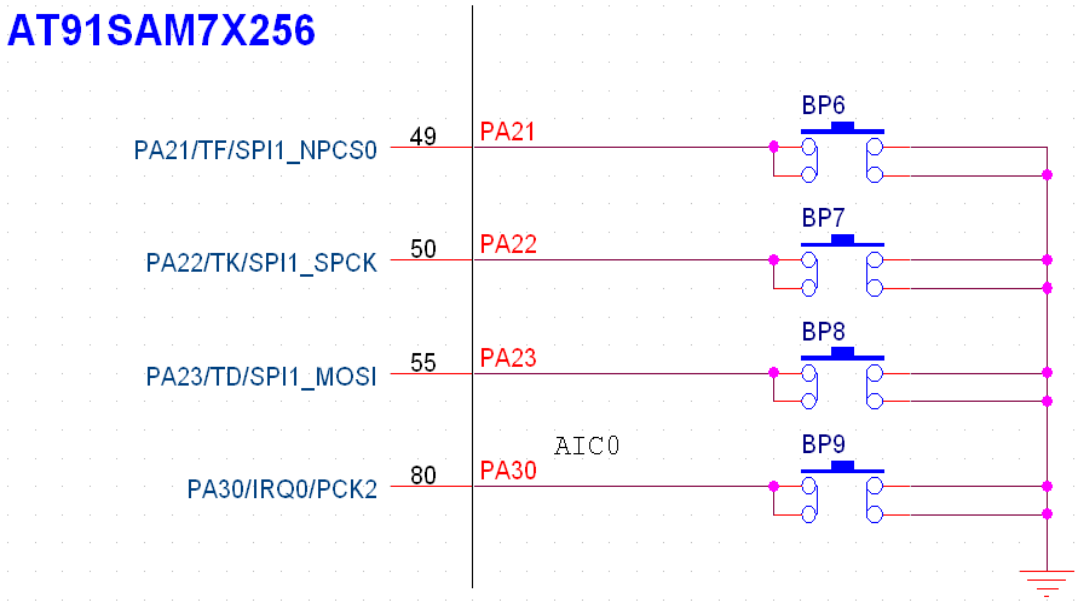
- LED 출력 부분

PORT B 출력 값은 “0” 로 Set 될 때, LED 가 점등(ON)되도록 설계되었으며, SAM7X256 의 경우 Output 으로 설정되면 기본적으로 “ 0 ” 값을 출력해준다.



- Switch 입력 부분

PORT A 입력 값은 “0” 이 입력될 때, Switch 가 “Active ”, “1”이 입력되면 “ Negative ”된다. 이는 Input 으로 설정된 PIO Port A 가 High 신호를 출력하기 때문이다. (PA30 의 경우, External IRQ 를 직접 제어하기 위해 별도로 구성되었다.)



5.3 MV-7X256 PIO 학습 예제

1) Key 를 눌렀을 때 해당 Key에 대응하는 LED가 켜지고, Key 를 떼면 LED 꺼지는 것을 보여라.

lib_AT91SAM7X256.h

```
1  //*  Enable peripheral clock
2  __inline void AT91F_PMC_EnablePeriphClock (
3          AT91PS_PMC pPMC, //  \ arg pointer to PMC controller
4          unsigned int periphIds)   //  \ arg IDs of peripherals
5  {
6          pPMC->PMC_PCER = periphIds;
7  }
8
9  //*  Enable PIO in output mode
10 __inline void AT91F_PIO_CfgOutput(
11         AT91PS_PIO pPio,              //  \ arg pointer to a PIO controller
12         unsigned int pioEnable)       //  \ arg PIO to be enabled
13 {
14         pPio->PIO_PER = pioEnable; // Set in PIO mode
15         pPio->PIO_OER = pioEnable; // Configure in Output
```

```
16  }
17
18  //*   Set to 1 output PIO
19  __inline void AT91F_PIO_SetOutput(
20          AT91PS_PIO pPio,          // \ arg   pointer to a PIO controller
21          unsigned int flag)        // \ arg   output to be set
22  {
23          pPio->PIO_SODR = flag;
24  }
25
26  //*   Set to 0 output PIO
27  __inline void AT91F_PIO_ClearOutput(
28          AT91PS_PIO pPio,          // \ arg   pointer to a PIO controller
29          unsigned int flag)        // \ arg   output to be cleared
30  {
31          pPio->PIO_CODR = flag;
32  }
```

IAR Compiler 는 별도의 lib_AT91SAM7X256.h 가 제공되는데, 사용자가 레지스터 번지를 매번 확인하고, 직접 레지스터를 건드려야만 하는 불편함이 없도록, C 기반에서 편리하게 사용 할 수 있도록 구성된 기본적인 일종의 AT91SAM7X256 Library 함수이다.

__inline void AT91F_PMC_EnablePeriphClock() // Enable peripheral clock

PMC(Power Management Controller)의 PMC_PCER (Peripheral Clock Enable Register) 를 Set 하여 Clock 을 Enable 시킨다.

Ex) AT91F_PMC_EnablePeriphClock (AT91C_BASE_PMC, 1 << AT91C_ID_PIOA) ;

참고로, "AT91C_ID_PIOA" 의 경우, ATMEL 에서 Peripheral 디버깅 편의를 위해서 각각의 Peripheral 별로 지정된 Peri ID (Peripheral Identifier) 가 주어지게 된다.

자세한 사항은 DATASheet 에서 " Pheripheral" 편을 참고하도록 한다.

10.2 Peripheral Identifiers

The AT91SAM7X512/256/128 embeds a wide range of peripherals. Table 10-1 defines the Peripheral Identifiers of the AT91SAM7X512/256/128. Unique peripheral identifiers are defined for both the Advanced Interrupt Controller and the Power Management Controller.

Table 10-1. Peripheral Identifiers

Peripheral ID	Peripheral Mnemonic	Peripheral Name	External Interrupt
0	AIC	Advanced Interrupt Controller	FIQ
1	SYSC[1]	System Controller	
2	PIOA	Parallel I/O Controller A	
3	PIOB	Parallel I/O Controller B	
4	SPI0	Serial Peripheral Interface 0	

__inline void AT91F_PIO_CfgOutput() // Enable PIO in output mode

각각의 Peripheral 을 PIO Mode 과 Output 모드로 설정해준다.

- PIO_PER : PIO Enable Register
- PIO_OER : Output Enable Register

Ex) AT91F_PIO_CfgOutput(AT91D_BASE_PIO_LED, AT91B_LED_MASK) ;

__inline void AT91F_PIO_SetOutput() // Set to 1 output PIO

원하는 PIOA/B의 특정 Bit 를 Set (1:Active) 시켜준다.

Ex) AT91F_PIO_SetOutput(AT91D_BASE_PIO_LED, AT91B_LED_MASK) ;

__inline void AT91F_PIO_ClearOutput() // Set to 0 output PIO

원하는 PIOA/B의 특정 Bit 를 Set (O:Negative) 시켜준다.

Ex) AT91F_PIO_ClearOutput(AT91D_BASE_PIO_LED, AT91B_LED_MASK) ;

__inline unsigned int AT91F_PIO_GetInput() // Return PIO input value

원하는 PIOA/B PIN I/O 값(PIO_PDSR : Pin Data Status Register)을 읽어 온다.

Ex) AT91F_PIO_GetInput(AT91D_BASE_PIO_SW);

AT91SAM7X256.h

```
1  // PIO DEFINITIONS FOR AT91SAM7X256
2  #define AT91C_PIO_PB21        ((unsigned int) 1 << 21)
3  #define AT91C_PIO_PB22        ((unsigned int) 1 << 22)
4
5  // BASE ADDRESS DEFINITIONS FOR AT91SAM7X256
6  #define AT91C_BASE_PMC        ((AT91PS_PMC)  0xFFFFFC00)
```

```
 7                                                    // (PMC) Base Address
 8  #define AT91C_BASE_PIOB       ((AT91PS_PIO)  0xFFFFF600)
 9                                                    // (PIOB) Base Address
10
11  // PERIPHERAL ID DEFINITIONS FOR AT91SAM7X256
12  #define AT91C_ID_PIOA    ((unsigned int)  2) // Parallel IO Controller A
13
```

AT91SAM7X-EK.h

```
 1  /*------------------*/
 2  /* LEDs Definition       */
 3  /*------------------*/
 4  #define AT91B_LED1           (1<<19)                 // AT91C_PIO_PB19
 5  #define AT91B_LED2           (1<<20)                 // AT91C_PIO_PB20
 6  #define AT91B_LED3           (AT91C_PIO_PB21)        // AT91C_PIO_PB21
 7  #define AT91B_LED4           (AT91C_PIO_PB22)        // AT91C_PIO_PB22
 8  #define AT91B_NB_LEB         4
 9  #define AT91B_LED_MASK
10                        (AT91B_LED1|AT91B_LED2|AT91B_LED3|AT91B_LED4)
11  #define AT91D_BASE_PIO_LED      (AT91C_BASE_PIOB)
12  #define AT91B_POWERLED              (1<<25)        // PB25
13
14  /*---------------------------------*/
15  /* Button Switch Position Definition   */
16  /*---------------------------------*/
17  #define AT91B_SW1            (1<<21)     // PA21
18  #define AT91B_SW2            (1<<22)     // PA22
19  #define AT91B_SW3            (1<<23)     // PA23
20  #define AT91B_SW4            (1<<30)     // PA30
21
22  #define AT91B_SW_MASK_MV   (AT91B_SW1|AT91B_SW2|AT91B_SW3)
23  #define AT91D_BASE_PIO_SW   (AT91C_BASE_PIOA)
```

>> ATSAM7X256 ECU board 제어를 위해서 사용된 사용된 헤더 파일이다.

main.c

```
1  #define __inline inline
2  #include "AT91SAM7X-EK.h"
3  #include "include/AT91SAM7X256.h"
4  #include "include/lib_AT91SAM7X256.h"
5
6  int main()
7  {   //* Begin
8     unsigned long InputSW;
9
10 // First, enable the clock of the PIOA
11 AT91F_PMC_EnablePeriphClock ( AT91C_BASE_PMC, 1 << AT91C_ID_PIOA ) ;
12 // Enable PIOB in output mode
13 AT91F_PIO_CfgOutput( AT91D_BASE_PIO_LED, AT91B_LED_MASK ) ;
14 // Clear(Set) the LED's.
15 AT91F_PIO_SetOutput( AT91D_BASE_PIO_LED, AT91B_LED_MASK ) ;
16 // Clear(Clear) the LED's.
17 AT91F_PIO_ClearOutput( AT91D_BASE_PIO_LED, AT91B_LED_MASK ) ;
18
19      while(1)
20      {   // PIOA 의 PDSR 을 계속 확인한다.
21      InputSW = ~(AT91F_PIO_GetInput(AT91D_BASE_PIO_SW));
22
23      If ( InputSW & AT91B_SW_MASK_MV)          // Switch 1/2/3 입력될 경우
24        AT91F_PIO_ClearOutput( AT91D_BASE_PIO_LED, InputSW>> 2 );
25
26           else if (InputSW & AT91B_SW4_MV )      // Switch 4 입력될 경우
27                AT91F_PIO_ClearOutput( AT91D_BASE_PIO_LED, AT91B_LED4 ) ;
28
29      else                                        // Switch 입력없을 경우
30           AT91F_PIO_SetOutput( AT91D_BASE_PIO_LED, AT91B_LED_MASK ) ;
31      } // End of While
32 }  // End of Main
```

1행 : inline 명령어를 __inline 으로 다시 정의한다. (기존 명령어 변경으로 인한)

2~4행 : 앞에서 소개된 헤더 파일을 정의한다.

11행 : Switch 입력으로 사용 될 PIOA 에 Clock 을 Enable 시켜준다.

13행 : PIOB 에서 LED 용으로 사용될 Peripheral 들을 Output mode 로 설정해 준다.
참고로, 현재 설정된 LED line (PB19 ~PB20) 은 GPIO 로만 동작되도록 기본 설정되어 있기 때문에 "PIO 설정" 은 해 줄 필요가 없다.

15~17행 : LED 사용 전에 테스트를 실시한다. (Set / Clear)

21행 : AT91D_BASE_PIO_SW PIN I/O 입력 값(SW1~4)을 확인한다.
(PULL-UP 설정에 따라서 기본적으로 High 값이 입력되지만, 수월한 비교를 위해서 입력된 PortA PIN I/O 값에 반전을 하여 InputSW 에 저장한다.)

23행 : InputSW 값이 Switch1/2/3 (PIOA 21/22/23) 인지 여부를 확인한다.

24행 : 만약 입력된 값이 Switch1/2/3 이라면, 대응하는 LED 를 출력해준다.
LED 와 SW 의 Port 번호가 틀려서 2bit shift 가 필요하다.

26행 : InputSW 값이 Switch4 (PIOA 30) 인지 여부를 확인한다.

27행 : 만약 입력된 값이 Switch4 이라면, 대응하는 AT91B_LED4 를 출력해준다.

29행 : InputSW 값이 발생하지 않을 경우.

30행 : LED 1/2/3/4 를 SET (OFF) 시켜준다.

참고로, 모든 I/O 는 Default Pull-up enable 설정되도록 되어 있으므로, Pull-up 에 대햇서 아무런 설정이 없다 하더라도 걱정할 필요가 없다.

27.5.1 Pull-up Resistor Control

Each I/O line is designed with an embedded pull-up resistor. The pull-up resistor can be enabled or disabled by writing respectively PIO_PUER (Pull-up Enable Register) and PIO_PUDR (Pull-up Disable Resistor). Writing in these registers results in setting or clearing the corresponding bit in PIO_PUSR (Pull-up Status Register). Reading a 1 in PIO_PUSR means the pull-up is disabled and reading a 0 means the pull-up is enabled.

Control of the pull-up resistor is possible regardless of the configuration of the I/O line.

After reset, all of the pull-ups are enabled, i.e. PIO_PUSR resets at the value 0x0.

5.4 MV-7X256 PIO 실습 예제

1. 모든 **LED** 가 **ON** 되어 있는 상태에서 **Switch** 값 입력이 들어오면 해당 **LED**를 **OFF** 한다**.**

10. SAM7X256 Startup code

10.1 MV7X256 Startup Code 이해

ARM과 같은 32bit 프로세서에서 프로그램을 개발하는 경우 대부분 C 언어를 사용한다. 그리고, C 언어는 기본적으로 Stack을 사용하게 된다. 그러므로 사용하게 될 스택에 대해서 초기화되어져야 하며, 더불어 사용하게 될 임베디드 시스템에 대한 기본적인 초기화 작업이 선행되어야 할 것이다. 이러한 작업을 하는 코드를 Startup code 라고 한다.

기본적으로 초기화 작업으로는 Exception 처리, Memory System, Stack, 전역변수 등의 초기화가 있으며, 이러한 작업을 수행하지 않을 경우에는, C 소스상에서 서브루틴의 호출이나 전역변수의 사용이 적절히 동작하지 않을 것이다.

물론, MV7X256 EDU Board 와 같은 OS 를 사용하지 않는 Firmware Program 일지라 하더라도 Core 에 관련된 혹은 H/W 시스템에 관한 적절한 Startup Code 가 필요하게 된다.

이러한 Startup code의 중요함에도 불구하고 대부분의 사용자가 이를 간과하는 이유는 몇 가지 사용상의 어려운 점 때문으로 보여진다. 첫 번째, 하드웨어 시스템을 전체적으로 파악해야 한다는 점과 두 번째, 이러한 설정을 어셈블리 기반으로 구성해야 하는 Startup code 의 특성이다.

따라서, 이장에서는 MV7X256 EDU board (ATMEL SAM7X256 MCU)의 Startup code 의 분석을 통하여, 사용자의 이해를 돕고자 한다.

10.1 MV7X256 Startup Code 분석

여기서 사용된 예제들은 MV7X256 EDU board 는 기준으로 작성되었으며, AT91SAM7X256 EDU Board 와 IAR EWARM Development Tools(Compiler) 4.11A 를 기준으로 작성하였다. C-startup 순서 예제는 AT91 Software Package 에 포함되어 있다.

EWARM Compiler 기반 MV7X256 EDU Board 는 아래와 같이 Startup code가 구성된다.

1. Cstartup.s79
2. Cstartup_SAM7.C
3. AT91SAM7X256.xcl
4. SAM7_Flash.mac
5. SAM7_RAM.mac
6. FlashAT91SAM7X.mac
7. FlashAT91SAM7x.d79

여기서 Startup code 는 크게, 두 가지 분류로 나누도록 하겠다. 1~3 목록은 C 언어로 작성이 된 Application source가 호출되기 전까지의 수행에 필요한 소스들이고, 4~7번 목록은 Debugging 을 위한 Startup Code로써, 참고 용도로 활용하도록 한다.

< C Start-up 用>
- Cstartup.s79 : 어셈블리로 구성된 System Start-up code로써, Exception 처리, Memory System, Stack, 전역변수 등의 초기화 등을 담당하고 있다.
- Cstartup_SAM7.C : C-startup Code 중 일부를 사용하기 편하게 C code 로 작성하여 지원함으로써, AT91SAM7X256 MCU 의 Master/PLL clock 설정을 편하게 변경할 수 있도록 하였다.
- AT91SAM7X256.xcl : Xlink Command File 로써, ROM/RAM Memory 의 Start/End 번지 정의하며, Stack/Interrupt Vector size 등을 설정할 수 있다.

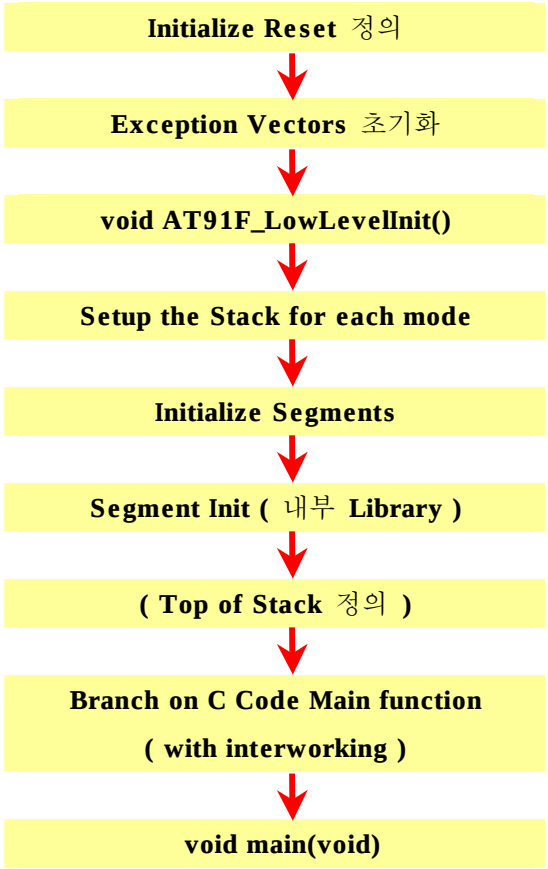
< C-Spy Start-up 用>
- SAM7_Flash.mac : Flash Debugging 용 Macro file (C-Spy 용)
- SAM7_RAM.mac : RAM Debugging 용 Macro file (C-Spy 용)
- FlashAT91SAM7X.mac : Flash Debugging 용 Macro file (FlashLoader 용)
- FlashAT91SAM7x.d79 : Flashloader 용 image download 실행 파일

10.1.2 Cstartup.s79

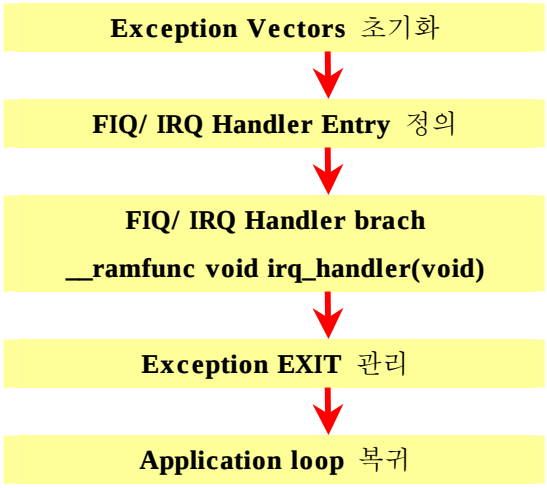
모듈성과 이식성을 위한 목적을 위하여 AT91SAM7S ARM-based Microcontroller 대부분 Application Code 가 " C " 로 작성이 되고 있다.
하지만, Startup Code 작성을 위해서는 ARM Processor 의 초기화 뿐만 아니라, 각각의 Peripheral에 대한 정확한 레지스터 사용(Register Architecture)과 Memory Mapping 진행(Processor)을 필요로 한다. 이러한 이유로 인해서 C Start-up Code 는 Assembler 를 사용하게 된다.

Reset이 된 **MCU**는 **Memory 0x0** 번지로 분기하여 아래의 같이 **Startup code**를 실행한다.

기본 실행 코드 **(C branch), Reset** 실행 코드 **(C-branch)** 등... **Irq/fiq** 실행코드,



< 전원 인가 시/ **Reset** 발생 시 >



< IRQ /FIQ Exception 발생 시 **>**

환경 초기화가 끝 난 후에, **application C Code** 를 수행한다.
다시 말하지만, **C-Start up file** 은 전원인가 후에 처음 실행되는 파일이며, **Reset vector**에

서 **application c code** 의 **main** 호출 전까지의 기본적인 **MCU**의 초기화 작업을 수행하게 된다.

1) Memory Entry point

Cstartup.s79

```
1  ;-  ------------------------------------------
2  ;- File source          : Cstartup.s79
3  ;- Object               : Generic CStartup
4  ;- 1.0 01/Sep/05 FBr    : Creation
5  ;- 1.1 09/Sep/05 JPP    : Change Interrupt management
6  ;------------------------------------------
7  #include "include/AT91SAM7X256_inc.h"
8
9  PROGRAM        RESET
10 RSEG   INTRAMEND_REMAP
11 RSEG   ICODE:CODE (2)
12 CODE32
13 ORG    0
```

7행 : ASM 용 Header File 적용한다. (C 용 Header File 의 경우 : AT91SAM7X256.h)
9행 : PROGRAM module 의 시작 선언한다. (Liker 를 위한 module 의 설정)
10행 : RSEG Segment 선언 후, 재배치가 가능한 Segment 영역으로 정의한다.
11행 : RSEG Segment 선언 후, 특정 Segment 'ICODE' 정의한다.
(C-startup 과 Exception Code 영역에 정의)
12행 : Reset 이후의 모든 명령어는 32-bit ARM mode 명령어를 사용하도록 정의한다.
13행 : 0x0 번지부터 Program module 이 시작되도록 정의한다.
(Reset vector Address 와 동일하게 잡아 주어야 한다.)

Note : 명령어 설명
- RSEG : Begins a relocatable segment (Segment Control Section)
- PROGRAM : Begins a program module (Module control)
- ORG : Sets the location counter (Segment control)

2) Exception Vectors Initialize

0x0 ~ 0x1C 번지까지의 **Exception Vector** 영역으로 설정하며**,** 각각의 **Exception** 은 **label** 별로 구성되어 **Exception (Data about, Undefined instruction, IRQ, etc..)** 발생시**, Core** 는 즉시 **0x00** 번지부터 **0x1C** 번지까지 **8**개의 명령어 중 하나의 서브루틴으로 분기 **(Branch)** 하게 된다**.**

Cstartup.s79

```
1  ;-------------------------------------------------------------
2  ;- Exception vectors
3  ;-------------------------------------------------------------
4  reset
5                  B          InitReset            ; 0x00 Reset handler
6  undefvec:
7                  B          undefvec             ; 0x04 Undefined Instruction
8  swivec:
9                  B          swivec               ; 0x08 Software Interrupt
10 pabtvec:
11                 B          pabtvec              ; 0x0C Prefetch Abort
12 dabtvec:
13                 B          dabtvec              ; 0x10 Data Abort
14 rsvdvec:
15                 B          rsvdvec              ; 0x14 reserved
16 irqvec:
17                 B          IRQ_Handler_Entry    ; 0x18 IRQ
18
19 fiqvec:                                         ; 0x1c FIQ
```

19행 : fiqvec 는 특별한 Branch 함수가 씌어있지 않다. 이는 빠른 수행을 위해서 바로 아래에서 Exception Handler 를 수행하기 위해서 이다.

따라서 정확한 벡터 동작을 위해서는 반드시 **Branch** 명령어로 분기할 수 있도록 함수가 준비되어야 하며**,** 이렇게 분기되어 수행되는 함수를 "**Exception Handler**" 라고 말한다**.**

- Exception Vectors 소개

Exception	**Description**

Reset	**Reset Exception** 은 **Processor** 전원이 공급되거나, **Resetting** 에 의해서만 동작되며, **Reset Vector** 를 사용해서 **Software Reset** 도 가능하다.
Undefined Instruction	프로세서나 혹은 **Coprocessor** 가 분석할 수 없는 명령어가 발생되었을 경우 수행되는 **Exception**
Software Interrupt(SWI)	사용정의에 의해서 수행되는 인터럽트 명령어. **User Mode** 에서 실행이 가능하다. – 해석요망
Prefetch Abort	**Processor** 가 부정확한 번지의 명령어를 수행하려고 할 경우 발생된다.
Data Abort	**Prefetch abort** 와 비슷하나, 부정확한 번지의 **data** 를 저장 혹은 읽으려는 **(Data** 전송**)** 명령어를 시도했을 경우 발생한다.
IRQ	**Processor** 에 외부 인터럽트 요청이 있을 경우나, **CPSR** 의 **I bit**가 **clear** 될 경우 발생된다.
FIQ	**IRQ** 보다 빠른 응답시간을 요구할 경우 사용되며, 동작은 **Processor** 에 외부 **Fast** 인터럽트 요청이 있을 경우나, **CPSR** 의 **F bit**가 **clear** 될 경우 발생된다.

3) FIQ Handler Entry 정의

FIQ Handler Entry 는 함수 명 자체가 의미하는 것처럼, **Application C code** 의 **FIQ Handler**로 분기하기 위한 레지스터의 백업작업이 이루어지게 된다.

Cstartup.s79

```
1  fiqvec:                                          ; 0x1c FIQ
2  ;---------------------------------------------------------------
3  ;- Function              : FIQ_Handler_Entry
4  ;- Treatments            : FIQ Controller Interrupt Handler.
5  ;- Called Functions      : AIC_FVR[interrupt]
6  ;---------------------------------------------------------------
7
8  FIQ_Handler_Entry:
9
10 ;- Switch in SVC/User Mode to allow User Stack access for C code
11 ; because the FIQ is not yet acknowledged
12
```

```
13  ;- Save and r0 in FIQ_Register
14              mov         r9,r0
15              ldr         r0 , [r8, #AIC_FVR]
16              msr         CPSR_c,#I_BIT | F_BIT | ARM_MODE_SVC
17  ;- Save scratch/used registers and LR in User Stack
18              stmfd       sp!, { r1-r3, r12, lr}
19
20  ;- Branch to the routine pointed by the AIC_FVR
21              mov         r14, pc
22              bx          r0
23
24  ;- Restore scratch/used registers and LR from User Stack
25              ldmia       sp!, { r1-r3, r12, lr}
26
27  ;- Leave Interrupts disabled and switch back in FIQ mode
28              msr         CPSR_c, #I_BIT | F_BIT | ARM_MODE_FIQ
29
30  ;- Restore the R0 ARM_MODE_SVC register
31              mov         r0,r9
32
33  ;- Restore the Program Counter using the LR_fiq directly in the PC
34              subs        pc,lr,#4
```

1행 : FIQ exception 선언

8행 : FIQ handler Call name 선언

Fast interrupt Request 는, 빠른 수행을 위해서 별도의 분기(branch) 명령은 없이 Exception 지정과 함께 Handler 가 시작이 되도록 되어있다.

14행 : 분기를 위한 r0 를 r9 로 임시 저장.

15행 : 이미 r8에 저장되어 있는 AT91C_BASE_AIC (0xFFFF F000) 번지와 AIC_FVR (Fast Interrupt Vector Register) 를 합하여 r0 로 분기하도록 한다.

16행 : Supervisor mode 로 진입하면서, IRQ mode 와 FIQ mode 를 비활성화 시킨다.

18행 : 이전 Mode 의 레지스터를 백업한다.

21행 : pc 값을 lr 로 백업한다.

22행 : FIQ Handler 로 분기한다.

25행 : 백업해 두었던 레지스터 값을 복원한다.

28행 : FIQ Mode 로 분기한다. IRQ mode 와 FIQ mode 를 비활성화 시킨다.

30행 : 이전에 r9 백업해 두었던 r0 값을 다시 복원한다.

34행 : lr 에 저장된 원 복귀주소에서 4를 뺀 주소 값을 pc 에 저장하여, 분기하도록 한다.

Auto Mode change instruction " C " 에 의해서 SVC mode 로 재 분기하여 정상 프로그램 수행을 계속진행 한다.

4) IRQ Handler Entry 정의

IRQ Handler Entry 는 함수 명 자체가 의미하는 것처럼**, Application C code** 의 **IRQ Handler**로 분기하기 위한 레지스터의 백업작업이 이루어지게 된다**.**

Cstartup.s79

```
1  ;------------------------------------------------------------------
2  ;- Function              : IRQ_Handler_Entry
3  ;- Treatments            : IRQ Controller Interrupt Handler.
4  ;- Called Functions      : AIC_IVR[interrupt]
5  ;------------------------------------------------------------------
6  IRQ_Handler_Entry:
7
8  ;- Adjust and save LR_irq in IRQ stack
9       sub         lr, lr, #4
10      stmfd       sp!, {lr}
11
12 ;- Save r0 and SPSR (need to be saved for nested interrupt)
13      mrs         r14, SPSR
14      stmfd       sp!, {r0,r14}
15
16 ;- Write in the IVR to support Protect Mode
17 ;- No effect in Normal Mode
18 ;- De-assert the NIRQ and clear the source in Protect Mode
19      ldr         r14, =AT91C_BASE_AIC
20      ldr         r0 , [r14, #AIC_IVR]
21      str         r14, [r14, #AIC_IVR]
22
23 ;- Enable Interrupt and Switch in Supervisor Mode
24      msr         CPSR_c, #ARM_MODE_SVC
25
```

```
26  ;- Save scratch/used registers and LR in User Stack
27      stmfd         sp!, { r1-r3, r12, r14}
28
29  ;---------------------------------------------
30  ;- Branch to the routine pointed by the AIC_IVR
31  ;---------------------------------------------
32      mov           r14, pc
33      bx            r0
```

6행 : " IRQ_Handler_Entry " IRQ Exception handler call name 을 설정한다.

9행 : IRQ Exception 이 발생되면, lr 은 마지막으로 실행된 명령어에 8을 더한 값을 가리키게 된다. 반드시 lr – 4 를 해야지만, Exception Handler 처리 후 복귀한 정확한 주소 값을 얻을 수가 있다.

10행 : 9행에서 감소된 lr 를 sp 에 백업하도록 한다.

13행 : SPSR 을 백업하기 위해서 lr 레지스터를 사용하여 복사하고 있다.

(중첩 인터럽트 방식을 사용하기 위해서 SPR 을 범용 레지스터에 저장을 한다.)

14행 : r0 와 SPSR 를 SP 에 백업하도록 한다.

(r14 는 현재 IRQ mode 의 SPSR 값이 r0 는 뒤에서 사용 예정이므로 백업해둔다.)

19행 : AIC Base 번지 (0xFFFF F000)를 r14에 저장한다.

20행 : r14 + AIC_IVR 번지 (0x0100) = 0xFFFF F100 에 저장된 Interrupt vector address 값을 r0 에 저장한다. (IRQ Handler Entry 종료 후 분기를 위한 준비 과정이다.)

21행 : r14의 data를 [0xFFFF F100] 번지의 data에 저장한다.

24행 : Supervisor mode로 변환한다.

27행 : Mode 변환 후, 레지스터를 백업한다. (r0 는 분기를 위해 백업하지 않는다.)

32행 : 현재 PC 의 값을 lr 로 백업한다.

33행 : IRQ Handler 로 분기한다.

IRQ Handler Entry 모드 IRQ Handler를 수행하기 위한 Entry 함수이지만, 이미 IRQ Mode (CPSR Mode set bit : 0b10010) 로 변경된 상태에서 진행이 되며, IRQ Handler Entry 에서 IRQ Handler 로 분기하기 전에 이전에 동작모드 (MV7X256의 경우 Supervisor Mode) 로 다시 Mode 가 변경된다. (사실 이는 다른 ARM core MCU 들과 틀리다.)

```
ldr     r14, =AT91C_BASE_AIC     // offset AT91C_BASE_AIC = 0xFFFF F000
ldr     r0 , [r14, #AIC_IVR]      // AIC_IVR = 0x0000 0100
bx      r0                        // R0 ( = 0xFFFF F100 ) 으로 이동
```

R0에서 R12는 Banked register 가 아니다. 이것은 레지스터들의 모든 mode 에서(IRQ mode 제외) 공유된다는 의미이다. R0 ~ R3, R12 는 ARM C 호출 편의상 Scratch Register 로 설정되었으며, 이 레지스터들은 반드시 모드분기 전에 우선적으로 저장이 되어야만 한다.

5) Exception EXIT 관리

IRQ Handler 수행을 마친 후, 다시 IRQ Handler entry 로 재 분기하여 IRQ Exception mode 수행을 마치게 된다.

Cstartup.s79

```
1  ;- Restore scratch/used registers and LR from User Stack
2      ldmia         sp!, { r1-r3, r12, r14}
3
4  ;- Disable Interrupt and switch back in IRQ mode
5      msr           CPSR_c, #I_BIT | ARM_MODE_IRQ
6
7  ;- Mark the End of Interrupt on the AIC
8      ldr           r14, =AT91C_BASE_AIC
9      str           r14, [r14, #AIC_EOICR]
10
11 ;- Restore SPSR_irq and r0 from IRQ stack
12     ldmia         sp!, {r0,r14}
13     msr           SPSR_cxsf, r14
14
15 ;- Restore adjusted  LR_irq from IRQ stack directly in the PC
16     ldmia         sp!, {pc}^
```

2행 : IRQ Handler 로 분기하기 전, 백업해두었던 레지스터를 다시 복원한다.

5행 : Interrupt Disable 후, IRQ Mode로 변경한다.

8행 : AIC Base 번지 (0xFFFF F000)를 r14에 저장한다.

9행 : 임의의 data 를 EOICR (End Of Interrupt Command Register :AIC Register)에 저장함으로써, Interrupt mode 를 종료한다.

12행 : 백업해 두었던 r0/r14 를 다시 복원한다.

13행 : r14 에 저장해 두었던, SPSR_irq 값을 다시 불러온다.

16행 : PC 값을 복원한 뒤, ^ 심볼을 통하여 spsr 값을 cpsr 로 저장하게 된다.
이로써, 원래의 동작 User(SVC) mode 로 변환하게 된다.

6) Initialize Reset 정의

MCU 가 처음 Booting 되면 PC 값을 0x00 을 갖게 되며, Exception vectors 에서 다음 명령을 대기하게 된다. 프로그램을 수행하기 위한 아무런 준비가 되어 있지 않기 때문에 Reset 된 직후의 처음 명령은 Call name " InitReset " 으로의 분기 함으로써, MCU 를 구동하기 위한 sp 의 위치 설정과

Cstartup.s79

```
1  InitReset:
2
3  ;- Retrieve end of RAM address
4  __iramend EQU SFB(INTRAMEND_REMAP)        ;- Segment begin
5
6      EXTERN    AT91F_LowLevelInit
7      ldr       r13,=__iramend
8
9  ;- Temporary stack in internal RAM for Low Level Init execution
10     ldr       r0,=AT91F_LowLevelInit
11     mov       lr, pc
12     bx        r0                  ;- Branch on C function (with interworking)
```

1행 : InitReset Call name 설정을 한다.
4행 : __iramend 에 xcl 파일에 지정되어 있는 INTRAMEND_REMAP 을 대입한다.
__iramend = 0020FFFF (MV7x256 Flash Debugging 의 경우)
SFB 명령어
6행 : 외부정의 AT91F_LowLevelInit 를 선언한다.
7행 : RAM 마지막 번지를 sp번지로 지정하여 저장한다.
이렇게 update 된, sp 번지는 push 될 때 마다, 번지가 하나씩 감소하며 data가 저장된다.
10행 : Cstartup_sam7.C 의 Main 함수에 해당하는 AT91F_LowLevelInit 함수로 분기하기 위한 번지를 임시 저장한다.
11행 : 분기하기 전 현재 pc 값을 lr 에 저장한다.
이때, 만약 pc값이 0x94 일 경우, 0x9C 가 자동으로 저장되어 다음 분기지점을

12행 : bx 명령어의 " x "에 의해서 ARM mode 에서 THUMB mode 로 mode 가 변경된다. 이러한 명령어 " x "를 Core 상태 변환 명령어라고 하며, 이렇게 ARM 코드와 Thumb 코드를 함께 사용하는 것을 ARM-Thumb 인터워킹이라고 하며, BX 와 BLX 명령어 등에 의해서 실행이 된다.

7) Top of Stack 정의

각 Exception Mode 별로 사용하게 될 Stack Size/ Mode 설정/ 인터럽트 마스크 bit 세팅에 관한 사전 정의가 이루어진다.

Cstartup.s79

```
1  IRQ_STACK_SIZE      EQU     (3*8*4)
2                      ; 3 words to be saved per interrupt priority level
3  ARM_MODE_FIQ        EQU     0x11
4  ARM_MODE_IRQ        EQU     0x12
5  ARM_MODE_SVC        EQU     0x13
6  I_BIT               EQU     0x80
7  F_BIT               EQU     0x40
```

1행 : IRQ Exception 발생시에 사용하게 될 Stack 의 Size 설정.
본 예제소스는 IRQ Exception 만을 사용하기 때문에 다른 Stack Size 에 대한 정의는 이루어지지 않았다. 만약 별도의 Exception 을 사용할 경우에는 반드시 설정을 잡아 줘야만 한다. (Application C code 에서의 Stack Size 는 별도로 XCL 파일에서 설정 해준다.)

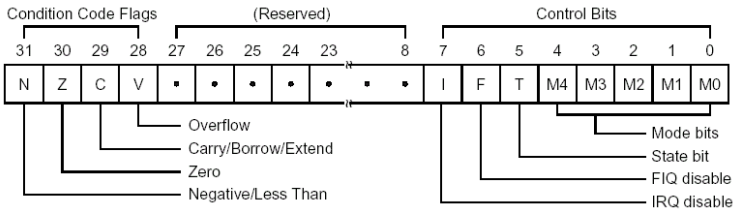
3행 : FIQ Processor Mode 로 설정하도록 정의한다.

4행 : IRQ Processor Mode 로 설정하도록 정의한다.

5행 : SVC Processor Mode 로 설정하도록 정의한다.

6행 : IRQ 인터럽트를 Disable 한다.

7행 : FIQ 인터럽트를 Disable 한다.



< Program Status Register >

M[4:0]	Mode
0b10000	User
0b10001	FIQ
0b10010	IRQ
0b10011	Supervisor
0b10111	Abort
0b11011	Undefined
0b11111	System

< Mode bit >

8) Setup the Stack for each mode

각각의 ARM mode 는 mode 별로 Stack pointer 를 가지고 있다. 따라서 각각의 모드에서 사용되는 Stack 들은 반드시 초기화가 필요하다.

Cstartup.s79

```
1     ldr     r0, =__iramend
2
3   ;- Set up Fast Interrupt Mode and set FIQ Mode Stack
4     msr     CPSR_c, #ARM_MODE_FIQ | I_BIT | F_BIT
5   ;- Init the FIQ register
6     ldr     r8, =AT91C_BASE_AIC
7
8   ;- Set up Interrupt Mode and set IRQ Mode Stack
9     msr     CPSR_c, #ARM_MODE_IRQ | I_BIT | F_BIT
10    mov     r13, r0                    ; Init stack IRQ
11    sub     r0, r0, #IRQ_STACK_SIZE
12
13  ;- Enable interrupt & Set up Supervisor Mode and set Supervisor Mode Stack
14    msr     CPSR_c, #ARM_MODE_SVC
15    mov     r13, r0
```

1행 : __iramend EQU SFB(INTRAMEND_REMAP) ;- Segment begin

INTRAMEND_REMAP 은 Internal RAM Memory 영역의 끝번지를 의미한다.

끝번지는 Stack Pointer 의 최상위 메모리번지를 의미하며, Set 될 때마다 번지가 감소될 것이다.

4행 : 현재 Mode를 FIQ Mode로 변경하며, Interrupt 와 Fast Interrupt 를 Disable 시킨다.
6행 : AT91C_BASE_AIC의 주소값 (0xFFFF F000) 을 R8 에 저장한다.
9행 : 현재 Mode를 IRQ Mode로 변경하며, Interrupt 와 Fast Interrupt 를 Disable 시킨다.
10행 : r0 값을 r13(Stack Point) 으로 복사한다.
11행 : r0 – (IRQ_STACK_SIZE) 값을 r0 에 저장하여, Stack Size 영역을 설정한다.
Stack 은 감소하게 된다. 다시 말하자면, internal SDRAM 의 가장 상위 번지에 위치해 있는 Stack pointer 는 data 가 저장 될수록 감소하게 된다.
14행 : 현재 Mode를 SVC Mode로 변경한다.
15행 : 스택번지 r13(SP) 로 설정하여 Stack Point 영역으로 설정하도록 한다.

일반적으로 Supervisor Mode(SVC) 와 User Mode 는 큰 사이의 Stack 용량을 갖게 되며, IRQ Mode 와 FIQ Mode 는 중간 정도의 Stack 용량, 나머지 Mode 들은 불과 몇 Byte 의 Stack 용량을 갖게 된다.

9) Initialize Segments

초기화된 변수나 **RAM code**의 초기값은 반드시 **Flash**에서 **RAM**으로 복사된다**.** 다른 모든 변수들도 또한 초기화 되어져야만 한다**.** " **__segment_init** " 함수는 **RAM Memory** 에 **Embedded Flash Code** 와 관련된 것들을 복사하며**, Data Segment** 의 초기화를 수행한다**.** **C** 초기화 또한**,** "**__segment_init**" 함수에서 진행되며**,** 이 함수는 **C-IAR Library** 에 포함되어 있으며**, Thumb / ARM Mode** 에서 모두 사용이 가능하다**.**

Cstartup.s79

```
1  EXTERN __segment_init
2      ldr     r0,=__segment_init
3      mov     lr, pc
4      bx      r0
```

1행 : 외부함수 __segment_init 를 선언한다.
2행 : r0 에 외부함수 __segment_init 의 호출 번지를 저장한다.
3행 : PC 레지스터의 값을 LR 에 로드한다. (복귀시 주소를 저장한다.)
4행 : __segment_init 로 ARM/Thumb Mode 를 변환하여 이동한다.

실제 소스 **: IAR_EWARM** 기본폴더 **\ ARM \ src \ lib \ dlib** 에서 확인 할 수 있다**.**

10) Branch on C Code Main function (with interworking)

컴파일러가 main() 함수를 컴파일 할 때, main 함수를 PUBLIC 선언이 자동으로 되며, main() 함수로 분기가 되면 C-startup 은 종료하게 된다.

Cstartup.s79

```
 1  EXTERN  main
 2      PUBLIC      __main
 3  ?jump_to_main:
 4      ldr  lr,=?call_exit
 5      ldr  r0,=main
 6  __main:
 7      bx  r0
 8  ?call_exit:
 9      End
10          b        End
```

1행 : 외부함수 main 를 선언한다. Application Source에서의 void main() 함수

2행 : " __main " 을 PUBLIC 선언한다. (Gloval 함수 선언)

3행 : " jump_to_main " call name 설정

4행 : " call_exit " call 주소를 lr로 로드한다.

5행 : " main " call 주소를 r0 로 로드한다.

6행 : " __main " call name 설정

7행 : Application source 중 main() 함수로 이동한다.

8행 : " call_exit " call name 설정

9행 ~ 10행 : 종료

11) Exception Vectors

사용하게 될 외부 handler Vector 설정을 한다.

Cstartup.s79

```
1  PUBLIC     AT91F_Default_FIQ_handler
2  PUBLIC     AT91F_Default_IRQ_handler
3  PUBLIC     AT91F_Spurious_handler
4
```

```
 5  CODE32        ; Always ARM mode after exeption
 6
 7  AT91F_Default_FIQ_handler
 8      b         AT91F_Default_FIQ_handler
 9
10  AT91F_Default_IRQ_handler
11      b         AT91F_Default_IRQ_handler
12
13  AT91F_Spurious_handler
14      b         AT91F_Spurious_handler
15
16      ENDMOD    ;- Terminates the assembly of the current module
17      END       ;- Terminates the assembly of the last module in a file
```

5행 : ARM Mode로 수행하도록 선언한다.

7~8행 : Application Source 에서의 FIQ Mode Exception Handler와 연동을 가능하게 해준다.

10~11행 : Application Source 에서의 IRQ Mode Exception Handler와 연동을 가능하게 해준다.

13~14행 : Application Source 에서의 Supervisor Mode Exception Handler와 연동을 가능하게 해준다.

10.1.2 Cstartup_SAM7.C

1) Embedded Flash Controller

내부 Flash 에 접근하기 위해서는 flash 의 동작속도에 비해서 Micro-controller core 의 Clock Speed 는 상당히 빠른 편이기 때문에 한번이상의 ' wait status '를 필요로 하게 된다. 즉, 설정된 일정한 시간동안 메모리의 접근을 기다리게 된다는 것이며, 이러한 ' wait status " 는 EFC (Embedded Flash Controller)에서 설정이 가능하다.

또한, Reset 후에 chip 은 매우 느린 내부 Clock(32KHz)으로 동작이 되기 때문에, 특별한 ' wait status '는 필요하지 않지만, chip 에서 지원되는 Full-Speed main Oscillator clock 을 사용하기 전에 반드시 ' wait status ' 의 설정이 마찬가지로 필요하다. 만일, 이러한 설정을 하지 않는 경우에는 Flash Memory의 접근이 불가능하다.

Cstartup_SAM7.C

```
1  #define  AT91C_MC_FWS_1FWS      ((unsigned int) 0x1 <<  8)
2                                  // (MC) 2 cycles for Read, 3 for Write operations
3
4  AT91C_BASE_MC->MC_FMR = AT91C_MC_FWS_1FWS;
5                                  // 1 Wait State necessary to work at 48MHz
```

4행 : 48MHz 동작할 경우 ' 1 wait status ' 의 Delay 만으로도 충분하다.
(EFC 의 Flash Mode Register(FMR) 에서 수정)

실제 ' wait status ' 는 micro-controller 의 동작 clock에 의해서 좌우되며, 좀더 많은 자료가 필요하다면, Datasheet 에서 " AC Electrical Characteristics section " 을 참고.

2) Main Oscillator and PLL

Reset 후에, chip 은 Slow Internal Clock(32KHz)으로 동작이 되기 때문에, Full Speed 동작을 원하거나, 일반적인 경우는 반드시 Main Oscillator 와 Phase Lock Loop (PLL) 설정을 해주어야만 한다. 이 두 가지 설정은 PMC (Power Management Controller) 에서 가능하다.

Cstartup_SAM7.C

```
1  // Enable Main Oscillator
2  AT91C_BASE_PMC->PMC_MOR =
3              (( AT91C_CKGR_OSCOUNT & (0x40 <<8) | AT91C_CKGR_MOSCEN ));
4  // Wait Main Oscillator stabilization
5   while(!(AT91C_BASE_PMC->PMC_SR & AT91C_PMC_MOSCS));
```

1행 : Oscillator Startup time 을 입력하고, PMC의 Main Oscillator Register(MOR)에서 MOSCEN bit를 enable 하여 Oscillator를 동작시킨다.

2행 : PMC Status Register의 MOSCS bit 가 Set 될 때 까지 기다린다.
이는, Main Oscillator를 활성화 한 뒤, 안정화 할 수 있는 시간을 필요로 한다.

Datasheet 에서 " DC characteristics " 를 확인하면 정확한 Oscillator startup time 을 계산할 수 있을 것이다. 그리고, AT91Sam7x 의 internal slow clock 은 RC oscillator에 해서 발생이 된다는 것을 명심하도록 한다.

2) PLL 설정

Oscillator 동작이 안정화되면, PLL 설정이 가능하다. PLL 설정은 입력된 클럭에 대해서 Divider 와 Multiplier 설정하는 과정을 의미한다.

이것은 Power Manager Controller의 PLL register (PLLR) 의 MUL 과 DIV 로 직접 설정할 수 있으며, 이 두 가지의 입력 값은 반드시 Main Oscillator Frequency (Input)과 희망하는 main Clock Frequency(Output) 값을 고려하여 설계되어져야 한다.

Cstartup_SAM7.C

```
1  ////////////////////////////////////////////////////////////////////////////////
2  // Set PLL to 96MHz (96,109MHz) and UDP Clock to 48MHz
3  ////////////////////////////////////////////////////////////////////////////////
4  #define        AT91C_CKGR_USBDIV_1                (0x1 << 28)
5  #define        AT91C_CKGR_OUT_1                   (0x1 << 14)
6  #define        AT91C_CKGR_PLLCOUNT                (0x3F <<  8)
7  #define        AT91C_CKGR_MUL                     (0x7FF << 16)
8  #define        AT91C_CKGR_DIV                     (0xFF <<  0)
9
10  AT91C_BASE_PMC->PMC_PLLR = AT91C_CKGR_USBDIV_1
```

```
11                                  | AT91C_CKGR_OUT_0
12                                  | AT91C_CKGR_PLLCOUNT
13                                  | (AT91C_CKGR_MUL & (72 << 16))
14                                  | (AT91C_CKGR_DIV & 14);
15
16  // Wait for PLL stabilization
17  while(!(AT91C_BASE_PMC->PMC_SR&AT91C_PMC_LOCK)
18  // Wait until the master clock is established for the case we already turn on the
19  PLL
20  while( !(AT91C_BASE_PMC->PMC_SR & AT91C_PMC_MCKRDY) );
```

25.9.9 PMC Clock Generator PLL Register

Register Name: CKGR_PLLR

Access Type: Read/Write

31	30	29	28	27	26	25	24
–	–	USBDIV		–	MUL		
23	22	21	20	19	18	17	16
MUL							
15	14	13	12	11	10	9	8
OUT		PLLCOUNT					
7	6	5	4	3	2	1	0
DIV							

Possible limitations on PLL input frequencies and multiplier factors should be checked before using the PMC.

< PMC Clock Generator PLL Register >

2행 : PLL Clock 은 96MHz 출력, UDP(USB Device Port) Clock 은 48MHz 설정 예정

10행 : PLL Clock Output 을 2로 나눠서 USB Clock으로 사용한다.

- **USBDIV: Divider for USB Clock**

USBDIV		Divider for USB Clock(s)
0	0	Divider output is PLL clock output.
0	1	Divider output is PLL clock output divided by 2.
1	0	Divider output is PLL clock output divided by 4.
1	1	Reserved.

11행 : PLL Clock Output 범위를 80MHz <= PLL Clock output Range <=160MHz

38.5 PLL Characteristics

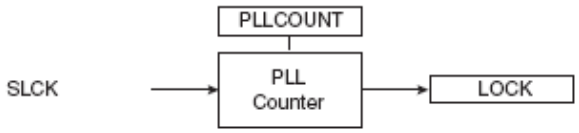
Table 38-12. Phase Lock Loop Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
F_{OUT}	Output Frequency	Field out of CKGR_PLL is:	00	80		160	MHz
			10	150		200	MHz
F_{IN}	Input Frequency			1		32	MHz
I_{PLL}	Current Consumption	Active mode				4	mA
		Standby mode				1	µA

Note: Startup time depends on PLL RC filter. A calculation tool is provided by Atmel.

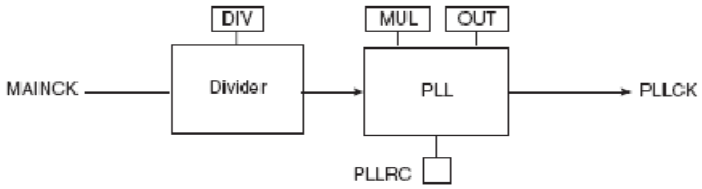
12행 : SLCK 의 (Slow Clock) Cycle 횟수를 지정함으로써, 지정된 PLLCOUNT 가 0으로 Decrement 되면, Lock bit 가 Set 된다. 따라서, Slow Clock Cycle 은 Lock bit 가 set 되기 전에 설정되어야 한다.

Figure 24-3. Divider and PLL Block Diagram



13행 : PLL Clock 주기(MUL)를 설정한다. PLL Clock = (Main Clock/DIV)*(MUL+1)

Figure 24-3. Divider and PLL Block Diagram



14행 : PLL Clock 주기(DIV)를 설정한다. PLL Clock = (Main Clock/DIV)*(MUL+1)

- **DIV: Divider**

DIV	Divider Selected
0	Divider output is 0
1	Divider is bypassed
2 - 255	Divider output is the selected clock divided by DIV.

17행 : PLLR의 값을 변경한 후에는, 반드시 PLL 이 Lock 이 걸릴 때까지 대기한다.

20행 : Master Clock 이 준비가 되면, MCKRDY bit (PMC_SR) 가 Set 된다.

PLL Clock = (Main Clock/DIV)*(MUL+1)

PLL output 을 내기 위한 계산법을 소개했다.

현재 MV7x256 EDU Board 를 예를 들자면

RC oscillator frequency range (kHz)	: 22<= f_{rc} <=42
Oscillator frequency range (MHz)	: 3<= f_{osc} <=20
Oscillator frequency on EK	: f_{osc} 18.432MHz //Input Frequency
Oscillator startup time	:1ms <= $t_{Startup}$ <= 1.4ms
Value for a 2ms startup	: OSCOUNT = (42000x0.0014)/8 = 8
F_{Output}	: (18.432 / 14) x 73 = 96.109 MHz

하지만, PLL 계산방법으로 DIV 와 MUL 값을 구한다는 것이 다소 까다로울 수 있기 때문에, Atmel 에서 자동으로 계산해주는 프로그램(PLL Calculator tool)을 제공하고 있기 때문에 [Oscillator frequency on EK] 와 [F_{Output}] 만으로도 쉽게 구 할 수가 있다.

Main Oscillator의 경우, PLL Startup time 은 반드시 설정되어야 하며, 또한, 해당 MCU 의 datasheet를 참조하여 DC characteristics 를 참고하면 계산이 가능하다는 점을 기억하도록 한다.

5) Prescaler 와 PLL Enable

마지막으로, main clock 의 Prescale 분주비를 설정하고, PLL Clock을 최종 사용유무 설정하도록 한다.

Prescale 값은 가장 처음에 설정되어야 하며, AC characteristics에 정의되어 있는 최고 동작 주파수 대역보다 높게 설정되는 것은 피해야만 한다. 아래의 예제처럼, 두개의 레지스터만 설정해주면 되며, 설정 후에는 반드시 Main clock 의 준비될 때까지 대기해야 한다.

여기서, MCU 는 Clock 구성에 따라서 main clock 과 PLL은 동작이 되도록 설정하였다.

Cstartup_SAM7.C

```
1  #define  AT91C_PMC_PRES_CLK_2      ((unsigned int) 0x1 <<  2)
2  // (PMC) Selected clock divided by 2
```

```
3   #define   AT91C_PMC_CSS_PLL_CLK      ((unsigned int) 0x3)
4   // (PMC) Clock from PLL is selected
5
6   AT91C_BASE_PMC->PMC_MCKR = AT91C_PMC_PRES_CLK_2;
7   // Wait until the master clock is established
8   while( !(AT91C_BASE_PMC->PMC_SR & AT91C_PMC_MCKRDY) );
9
10  AT91C_BASE_PMC->PMC_MCKR |= AT91C_PMC_CSS_PLL_CLK;
11  // Wait until the master clock is established
12  while( !(AT91C_BASE_PMC->PMC_SR & AT91C_PMC_MCKRDY) );
```

25.9.10 PMC Master Clock Register

Register Name: PMC_MCKR

Access Type: Read/Write

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
–	–	–	PRES			CSS	

< PMC Master Clock Register >

6행 : Processor Clock Prescaler 를 2 로 나눈다.

- **PRES: Processor Clock Prescaler**

PRES			Processor Clock
0	0	0	Selected clock
0	0	1	Selected clock divided by 2
0	1	0	Selected clock divided by 4
0	1	1	Selected clock divided by 8
1	0	0	Selected clock divided by 16
1	0	1	Selected clock divided by 32
1	1	0	Selected clock divided by 64
1	1	1	Reserved

8행 : Prescaler 설정 후, Main clock 동작 대기한다.

10행 : PLL Clock 을 Enable 시킨다.

- **CSS: Master Clock Selection**

CSS		Clock Source Selection
0	0	Slow Clock is selected
0	1	Main Clock is selected
1	0	Reserved
1	1	PLL Clock is selected.

11행 : Clock 설정 후, Main Clock 동작 대기한다.

3) Watchdog 설정

Watchdog peripherial 은 Default enable 설정되어 있어야 한다. 만약 Wtachdog 기능을 사용하지 않을 것이라면, 아래의 예제처럼 Watchdog Mode Register (WDMR)를 disable 시켜주면 된다.

Cstartup_SAM7.C

```
1  #define AT91C_WDTC_WDDIS          (0x1 << 15)
2
3  AT91C_BASE_WDTC->WDTC_WDMR = AT91C_WDTC_WDDIS;
4  // (WDTC) Watchdog Disable
```

3행 : Watchdog Timer Disable Bit 를 SET 시켜준다.

16.4.2 Watchdog Timer Mode Register

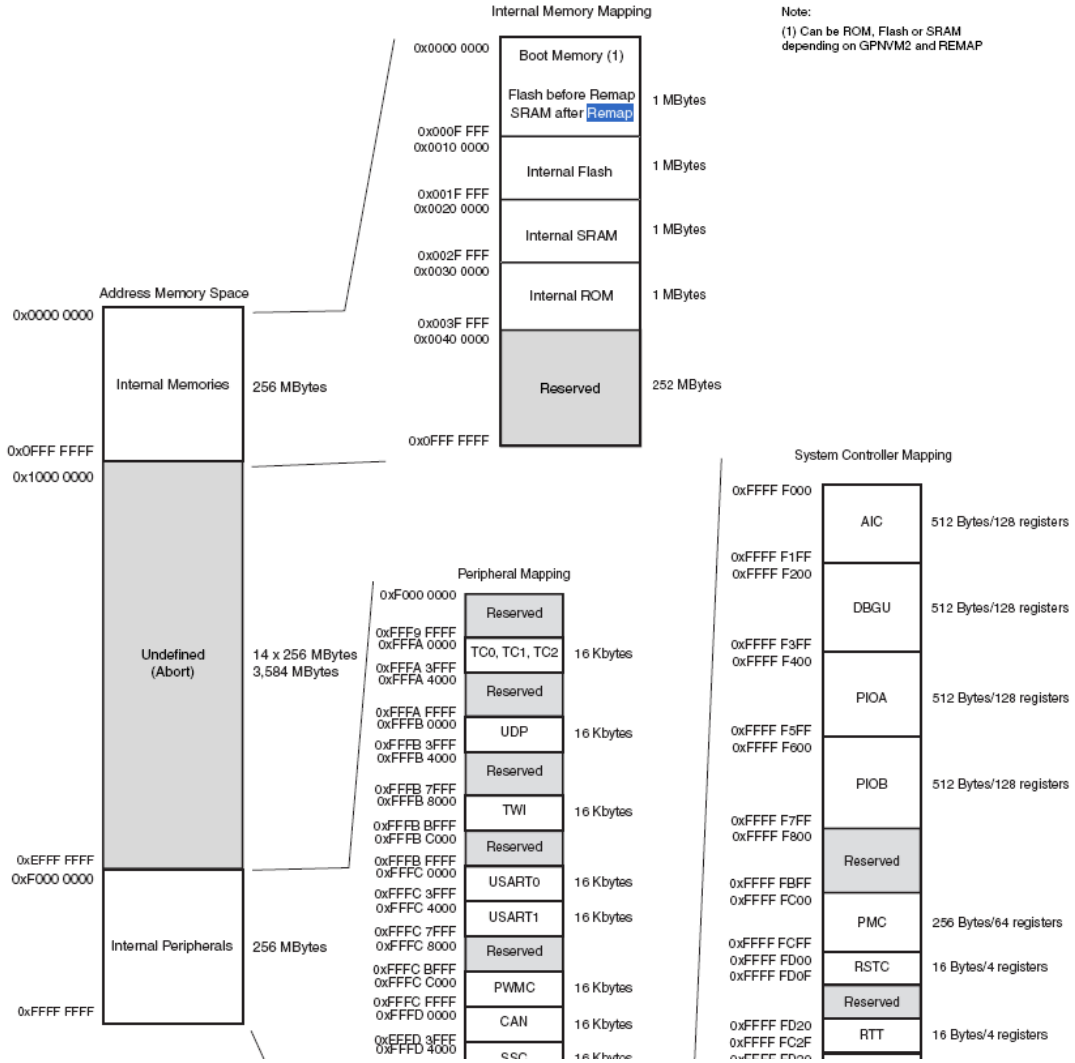
Register Name: WDT_MR

Access Type: Read/Write Once

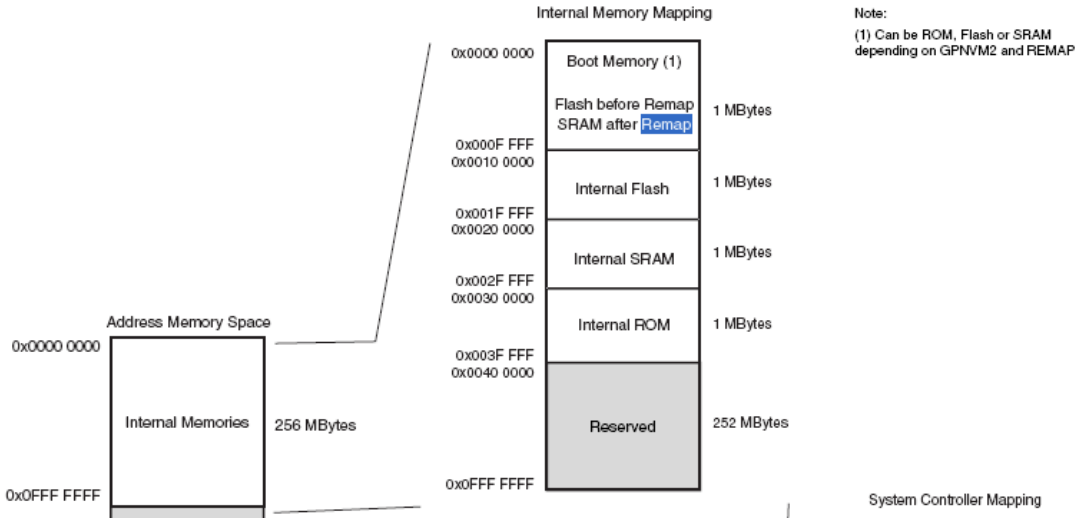
31	30	29	28	27	26	25	24
–	–	WDIDLEHLT	WDDBGHLT	WDD			
23	22	21	20	19	18	17	16
WDD							
15	14	13	12	11	10	9	8
WDDIS	WDRPROC	WDRSTEN	WDFIEN	WDV			
7	6	5	4	3	2	1	0
WDV							

< Watchdog Timer Mode Register : WTMR >

Figure 8-1. AT91SAM7X512/256/128 Memory Mapping



ATSAM7x 시리즈의 core 는 위와 같은 Memory mapping 구조를 갖는다.



혼돈이 되는 것은 바로 이 영역인데, 0x0000 0000 ~ 0x000F FFFF 영역은
Boot Memory 라고 사용되는 영역으로써, xcl 파일과 관련 레지스터 MC_RCR에 의해서 조정되는 Remapping 기능을 수행함에 따라서, 영역이 달리 사용될 수 있다.

Boot Memory(1)

Flash before Remap

SRAM after Remap

(1) Can be Rom, Flash or SRAM, it depending on GPNVM2 and REMAP

19.3.3 MC Flash Status Register

Register Name: MC_FSR

Access Type: Read-only

Offset: (EFC0) 0x68

Offset: (EFC1) 0x78

31	30	29	28	27	26	25	24
LOCKS15	LOCKS14	LOCKS13	LOCKS12	LOCKS11	LOCKS10	LOCKS9	LOCKS8
23	22	21	20	19	18	17	16
LOCKS7	LOCKS6	LOCKS5	LOCKS4	LOCKS3	LOCKS2	LOCKS1	LOCKS0
15	14	13	12	11	10	9	8
–	–	–	–	–	GPNVM2	GPNVM1	GPNVM0
7	6	5	4	3	2	1	0
–	–	–	SECURITY	PROGE	LOCKE	–	FRDY

- **GPNVMx: General-purpose NVM Bit Status** (Does not apply to EFC1 on the AT91SAM7X512.)

0: The corresponding general-purpose NVM bit is inactive.

1: The corresponding general-purpose NVM bit is active.

각각의 메모리 영역에 대해서 서술한 내용을 살펴보면,

1. Internal SRAM

Reset 이 걸리고 나서 Remap 명령이 수행되기 전.
SRAM 은 반드시 지정된 영역(0x0020 0000)에서만 수행이 가능하다.
하지만, Remap 이 수행된 후에는 SRAM 은 0x0 번지에서도 수행이 가능하다.
(The also becomes available at address 0x0)

Reset(0), Remap(0) > 0x0 에서도 Internal SRAM 영역 접근 가능

2. Internal ROM

모든 SAM7x 시리즈는 Internal Rom 을 갖고 있으며, 어떠한 때라도 Internal Rom영역은 0x0030 0000 이다.

3. Internal Flash

어떠한 경우라도 Flash 영역은 0x0030 0000 번지를 갖으며,
동시에, Reset 후에는 0x0 번지에서도 또한 접근이 가능하다.
단, CPNVM bit 2 가 Clear(0) 되어있으며, Remap 은 수행이 되지 않았을 경우.

Reset(0), Remap(x) > 0x0 에서도 Internal Flash 영역 접근 가능

4. Boot Memory (Internal Memory Area 0)

일반적으로 Internal Memory 영역 초기 32byte 는 ARM processor 의 Exception Vector 를 포함한다. (Reset Vector at address 0x0.)

Reset(0), Remap(x) > 0x0 에서도 Internal Flash 영역 접근 가능
Reset(0), Remap(0) > 0x0 에서도 Internal SRAM 영역 접근 가능

또한, internal SRAM(0x0020 0000) 영역이 internal Memory Area 0 (0x0000 0000) 영역으로 mapping 된다. 이는 단순히 copy 만 된 것이 아니라, 0x0 번지에서도 Internal SRAM 의 write/read 가 모두 가능하게도 된다.
(The internal SRAM at address 0x0020 0000 is mapped into internal Memory Area 0. The Memory into internal Memory Area 0 is accessible in both its original location and at address 0x0)

Remap의 이유는 모든 프로그램의 시작번지를 0x0 으로 함으로써, Flash debugging 과 RAM debugging 시에 나타나타날 수 있는 시작번지의 혼돈을 피하기 위해서 이다.

하지만, 알아둬야 할 것은 0x0 번지를 사용을 하더라도, remap 실행 여부에 따라서 실제는 다른 영역을 사용하고 있다는 것을 인식하고 있어야 할것이다.

Q) RAM debugging 이라는 의미는 실제 code를 ROM 이 아닌 RAM 에서 디버깅한다는 의미가 아닌가?? RAM 과 ROM 이 왜 번지가 동일한가???

Remap 전

Flash
Flash
SRAM
ROM

Remap 후

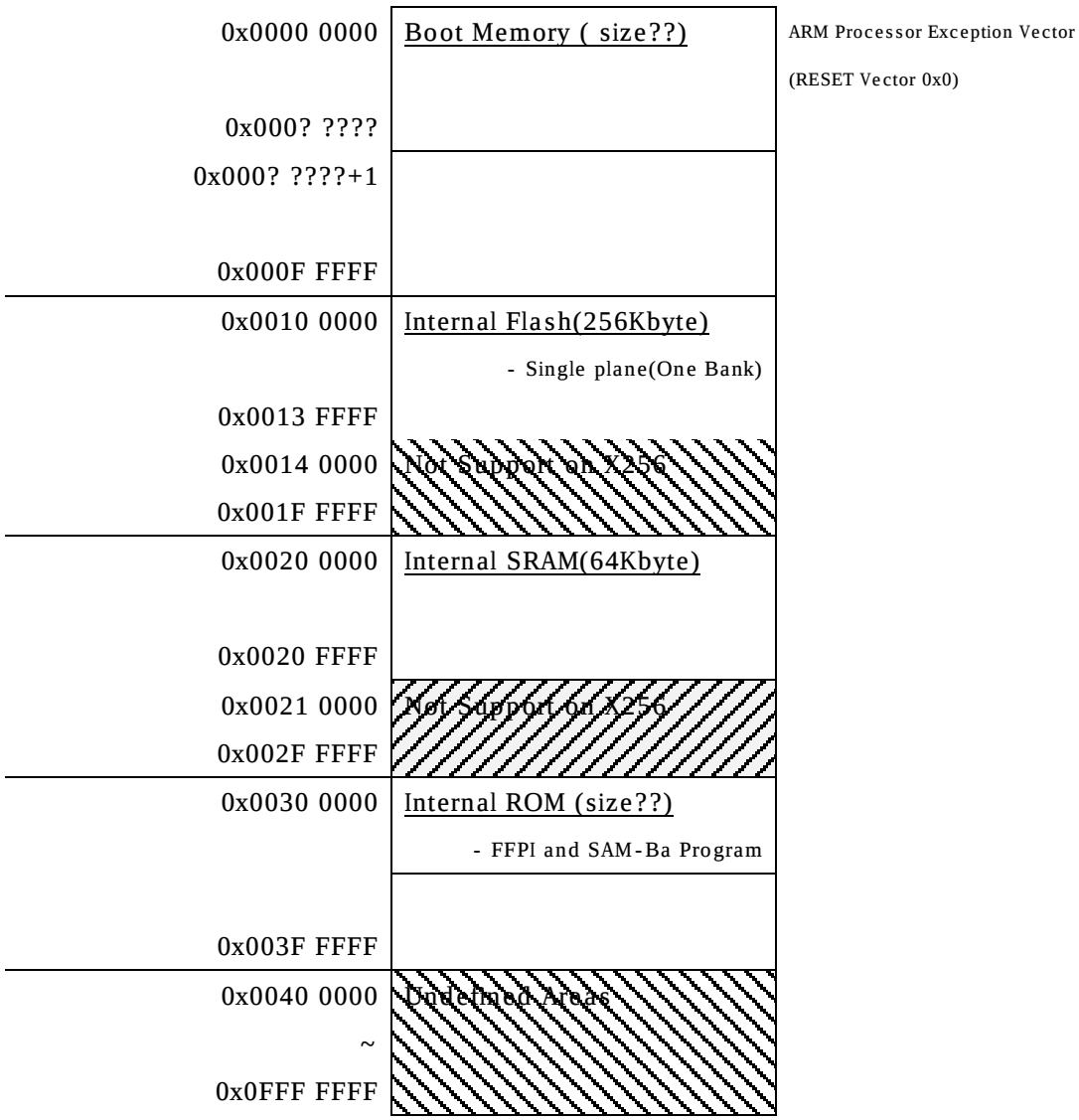
SRAM
Flash
SRAM
ROM

	IN .XCL file(Flash Debugging)		IN .XCL file(RAM Debugging)
Flash	0x0000 0000		0x0000 0000
SDRAM	0x0020 0000		0x0000 0000

// NOREMAP
ROMSTART
Start Address 0x0000 0000
RAMSTART
Start Address 0x0020 0000

// REMAP
RAMSTART
Start Address 0x0000 0000
ROMSTART
Start Address 0x0010 0000 // No matter.

Remap 후 (SAM7X256 경우)



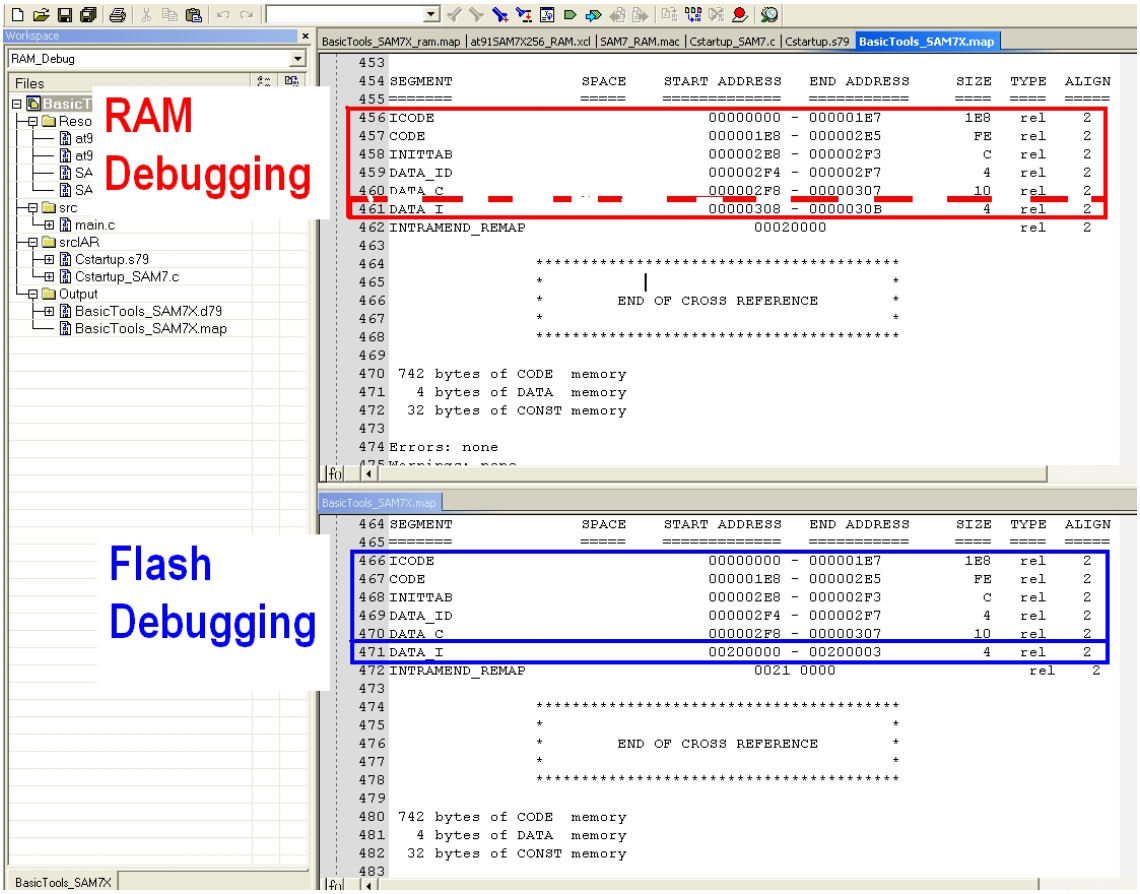
Remap 명령어를 실행시키기 전에, Internal Memory Area 0 는 on-chip Flash 로 Mapped 되어 있다. Remap 명령어를 실행시키기 후에는, Internal Memory Area 0 는 0x0020 0000번지의 internal SDRAM 으로 Mapped 되게 된다.

이렇게 Internal Memory Area 0 (0x0 번지)내에 mapped memory 는 실제 0x0020 0000 (internal SDRAM) 번지와 동시에 accessible 이 가능하다.

Remap Commnad :

실행 후에, internal SRAM 은 Internal Memory Area 0 을 통하여 accessed 가 가능하다.

As the ARM vectors(Reset, Abort, Data Abort, Prefetch Abort, Undefined Insruction, Interrupt and Fast Interrupt) 는 0x0 에서 0x20 번지까지 mapped 되며, Software 적으로 제어하여, exception vector 선언에 대한 재 정의가 가능하다.



Remapping 이라는 말은 ATMEL 7TDMI 에서 만 사용이 된다. 그 이유는 0x0000 0000 ~ 0x000F FFFF 영역(1Mbyte)의 Boot Memory 라는 녀석 때문이다.
이 녀석은 RAM 처럼 혹은, Flash 처럼 사용할 수도 있는 특이한 녀석이다. ??
어떻게 두 가지 물리적개념을 동시에 갖을 수 있느냐고 반문할 수 있을 것이다.
물론, 불가능하다. 여기서는 물리적으로 완전히 바뀌는 것이 단순히 Access point 역할만을 하기 때문에 가능하게 되는 것이다.

Boot Memory 는 최초 reset 부팅 당시에는 Flash 영역과 Accessing 되어 Flash 영역 즉 , 0x0010 0000 번지를 0x0000 0000 번지에서도 자유롭게 읽을 수가 있다.

하지만, MC remap Control Register의 RCB bit 를 set 해줌으로써, Flash Memory 를 Accessing 하던 Boot memory 는 SRAM Memory (0x0020 0000 번지)를 Access 하게 된다. 물론, 이때는 Read/write 가 가능해진다.

이렇게, boot Memory 의 Access point 를 변경해주는 것을 ATMEL ARM Core 에서는 Remapping 이라고 선언해서 사용하고 있으며, 사실은 일반적인 ARM 에서의 Remapping

하고는 차이를 보이는 것이다.

일반적으로, Remap 이라는 것은, Flash/NOR Memory 에 올라가 있는 OS등의 이미지(커널 이미지) SDRAM 으로 다시 부트로더에 의해서 다시 옮기는 수행을 의미한다.

- Ø Nand Flash 는 블록단위로 구성되어 있어서 SDRAM 에 비해서 처리 속도가 느리다. (Nand Flash 는 단순 저장기능만 갖기 때문에, remapping 후 사용되지 않는다. ?? 물론, OS 를 사용하는 ARM 기반일 경우에는..)
- Ø Application 프로그램의 주변 Task 들과의 유기적인 동작수행을 위해서.

그리고, 다른 MCU vendor사 ST 의 경우에는 Remapping 이라는 개념이 사용이 되고있다.
BOOTEN
BOOTT0
BOOTT1
설정을 하여, RAM/ Flash/ EXTMEM 접근가능.

참고로, 이 Remapping 이라는 것은 RAM debugging 시에만 사용이 되기 때문에, 무조건 해야하는 것은 아니다~~

앗.. 이게 가장 중요한 이야기인 것 같은 기분이 갑자기 든다.