

Linux 기초편

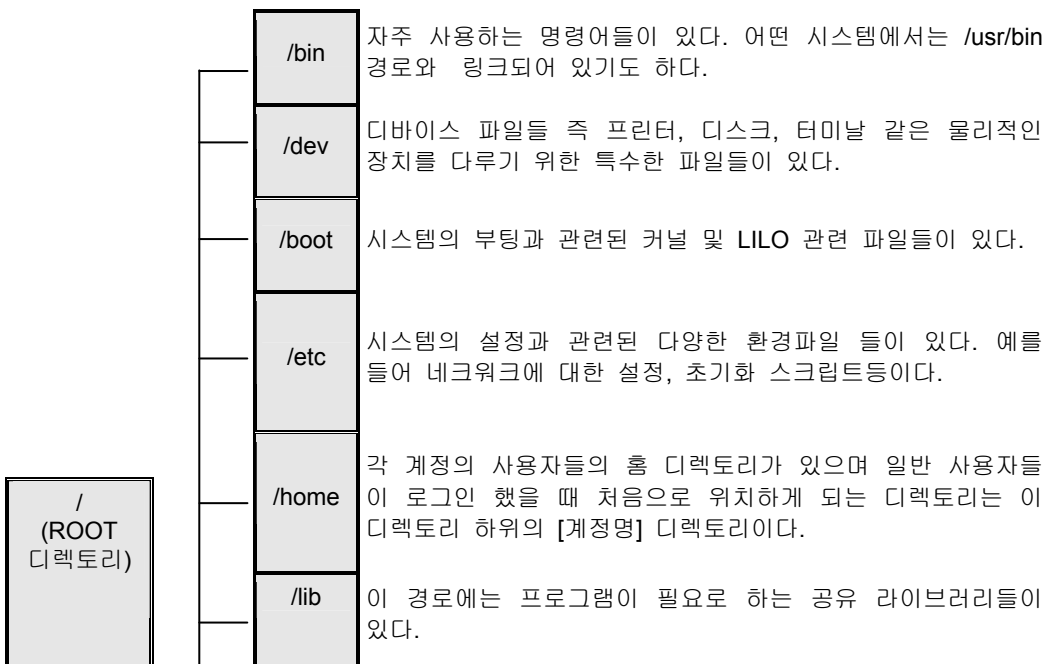
여기서 소개하는 명령들은 LINUX의 기본적인 명령군에 속하는 것이며, 이것들은 적어도 한번 이상은 실행시켜 보아 어떤 역할을 하는지 알아두어야 할 것이다. LINUX의 설치에 대해서는 각종 LINUX 배포판들이 존재하고 있으며, 이들은 해당 배포판의 홈페이지에서 자세히 설명해주고 있으므로 이를 참조하도록 하자.

이 책이 LINUX 기초에 관한 것이 아니라 임베디드리눅스에 대한 책임을 상기하면서 이 기본 명령어들에 대한 충분한 설명이 부족하더라도 이해하기 바란다.

2.1. 파일과 디렉토리 관련


LINUX 디렉토리는 "/" 로 표현되는 최상위 디렉토리(Root 디렉토리라 부른다)를 기반으로 그 하부에 서브디렉토리, 그 하부에 파일 및 서브디렉토리등의 구성의 트리구조(계층적구조)를 가진다. LINUX에서는 각 디렉토리마다 특정 파일들을 저장할 수 있도록 되어 있으며, 어떤 디렉토리에 어떤 종류의 파일들이 들어있는 가를 아는 것은 앞으로의 작업에 큰 도움이 된다.

아래 디렉토리와 그에 해당하는 내용들을 설명하고 있다. 시스템이나 사용자에 따라 디렉토리 구조가 조금씩 달라질 수는 있겠으나 큰 맥락에서는 거의 같다고 볼 수 있을 것이다.



	/mnt	임시 마운트용 디렉토리로 사용되며, 플로피나 시디롬 등의 자원을 이 디렉토리 아래 마운트하여 사용한다.
	/proc	메모리상에만 있는 가상 파일 시스템으로 시스템에서 운영되고 있는 프로세서들과 메모리 및 프로그램에 대한 정보를 담고 있는 파일들이 있다.
	/root	root 사용자 의 홈 디렉토리이다.
	/sbin	시스템 관리용 명령어들이 들어 있다.
	/tmp	임시저장용 디렉토리이다.
	/usr	공용으로 사용할 수 있는 명령파일이나 응용프로그램들이 존재한다. 대부분의 사용 명령들은 /bin, /sbin, /usr/bin/, /usr/sbin 디렉토리에 존재한다.
	/var	시스템 운영 중에 생성되는 각종 로그 파일들과 사용자의 메일등을 저장한다.

이상에서 LINUX 디렉토리의 종류와 그 내용에 대해서 살펴보았고 아래에서는 LINUX 디렉토리와 파일들을 다룰 수 있는 명령들을 설명한다.

명 령	내 용						
pwd	현재 작업중인 디렉토리 위치를 출력한다.						
cd	<p>작업 디렉토리를 이동한다.</p> <div> <p> 응용</p> <p>cd : 그냥 cd 명령만으로 홈디렉토리로 이동</p> <p>cd ~ : 홈 디렉토리 이동(틸트라고 부른다. 틸트 표시 "~"는 셸이 \$HOME환경변수의 값으로 대치하여 해석한다.)</p> <p>cd .. : 상위 디렉토리로 이동</p> <p>cd . : 현재 디렉토리로 이동</p> <p>cd - : 이전 디렉토리로 이동</p> </div>						
ls	<p>도스의 "dir"과 비슷한 기능의 명령어로서 파일 목록을 출력하며, 인수를 주지 않을 경우에는 현재 디렉토리내의 파일목록을 출력한다.</p> <table> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-l</td><td>파일에 대한 이름외에 좀더 자세한 정보 즉, 파일의 종류, 링크 수, 디렉토리, 소유그룹, 파일사이즈, 변경날짜등에 대한 정보를 출력한다.</td></tr> <tr> <td>-a</td><td>파일명이 "."로 시작하는 숨겨진 파일까지 출력한다..(UNIX 계열 시스템에서 숨겨진 파일들은 이름이 "."으로 시작한</td></tr> </table>	옵 션	설 명	-l	파일에 대한 이름외에 좀더 자세한 정보 즉, 파일의 종류, 링크 수, 디렉토리, 소유그룹, 파일사이즈, 변경날짜등에 대한 정보를 출력한다.	-a	파일명이 "."로 시작하는 숨겨진 파일까지 출력한다..(UNIX 계열 시스템에서 숨겨진 파일들은 이름이 "."으로 시작한
옵 션	설 명						
-l	파일에 대한 이름외에 좀더 자세한 정보 즉, 파일의 종류, 링크 수, 디렉토리, 소유그룹, 파일사이즈, 변경날짜등에 대한 정보를 출력한다.						
-a	파일명이 "."로 시작하는 숨겨진 파일까지 출력한다..(UNIX 계열 시스템에서 숨겨진 파일들은 이름이 "."으로 시작한						

	<table> <tr> <td></td><td>다.)</td></tr> <tr> <td>-t</td><td>변경된 파일들 중 최근에 변경된 것부터 출력한다..</td></tr> <tr> <td>-R</td><td>하위 디렉토리의 파일까지 출력한다..</td></tr> <tr> <td>-d</td><td>디렉토리 항목만 출력한다..</td></tr> </table>		다.)	-t	변경된 파일들 중 최근에 변경된 것부터 출력한다..	-R	하위 디렉토리의 파일까지 출력한다..	-d	디렉토리 항목만 출력한다..
	다.)								
-t	변경된 파일들 중 최근에 변경된 것부터 출력한다..								
-R	하위 디렉토리의 파일까지 출력한다..								
-d	디렉토리 항목만 출력한다..								
mkdir	<p>디렉토리를 생성하는 명령어로 아래의 -p 옵션을 주어 디렉토리를 설정하면 설정된 하부 디렉토리들이 없다면 모두 만들어 진다.</p> <table> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-p</td><td>현재 디렉토리에서 하위 디렉토리까지 생성한다. <code>mkdir -p /usr/src/arm</code></td></tr> </table>	옵 션	설 명	-p	현재 디렉토리에서 하위 디렉토리까지 생성한다. <code>mkdir -p /usr/src/arm</code>				
옵 션	설 명								
-p	현재 디렉토리에서 하위 디렉토리까지 생성한다. <code>mkdir -p /usr/src/arm</code>								
mv	<p>디렉토리나 파일을 옮기는 명령어이며 이름을 변경하는 경우에도 사용된다.</p> <table> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-- backup</td><td>복사하려는 파일과 같은 파일이 있다면, 덮어 씌울지 확인한 후 백업 파일을 원본 파일 뒤에 틸드(~)를 붙여서 생성하고 이동한다.</td></tr> <tr> <td>-v</td><td>작업진행내용 출력한다..</td></tr> <tr> <td>-f</td><td>겹쳐쓰기 여부를 묻지않고 강제로 이동한다. (조심해서 사용하자.)</td></tr> </table>	옵 션	설 명	-- backup	복사하려는 파일과 같은 파일이 있다면, 덮어 씌울지 확인한 후 백업 파일을 원본 파일 뒤에 틸드(~)를 붙여서 생성하고 이동한다.	-v	작업진행내용 출력한다..	-f	겹쳐쓰기 여부를 묻지않고 강제로 이동한다. (조심해서 사용하자.)
옵 션	설 명								
-- backup	복사하려는 파일과 같은 파일이 있다면, 덮어 씌울지 확인한 후 백업 파일을 원본 파일 뒤에 틸드(~)를 붙여서 생성하고 이동한다.								
-v	작업진행내용 출력한다..								
-f	겹쳐쓰기 여부를 묻지않고 강제로 이동한다. (조심해서 사용하자.)								
rmdir	<p>디렉토리를 지우는 명령어이며 이때 지우려는 디렉토리는 비어 있어야 한다. 만약 해당 디렉토리가 비워 있지 않다면 지울 수 없으며, 이때는 해당 디렉토리의 파일들을 모두 지운 후 이 명령어를 사용하여 최종 디렉토리를 지워야 하는데, 이런 과정 없이 하위 파일들까지 모두 지우고 싶다면 "rm -r" 명령으로 지우는 것이 사용하기 쉽다.</p> <table> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-p</td><td>디렉토리명에 적은 하위 디렉토리부터 상위 디렉토리까지 순서대로 삭제한다.</td></tr> <tr> <td>-v</td><td>디렉토리가 제거되는 작업진행내용 출력한다..</td></tr> <tr> <td>-f</td><td>겹쳐쓰기 여부를 묻지않고 강제로 삭제한다.</td></tr> </table>	옵 션	설 명	-p	디렉토리명에 적은 하위 디렉토리부터 상위 디렉토리까지 순서대로 삭제한다.	-v	디렉토리가 제거되는 작업진행내용 출력한다..	-f	겹쳐쓰기 여부를 묻지않고 강제로 삭제한다.
옵 션	설 명								
-p	디렉토리명에 적은 하위 디렉토리부터 상위 디렉토리까지 순서대로 삭제한다.								
-v	디렉토리가 제거되는 작업진행내용 출력한다..								
-f	겹쳐쓰기 여부를 묻지않고 강제로 삭제한다.								
cat	<p>이 cat 명령은 리다이렉션¹을 이용하여 간단한 파일을 만들 때나 파일의 내용을 볼 때 사용한다. 텍스트 파일`을 만들고, 보기위해 서는 cat 명령 이외에도 more, touch명령들 또는 vi 응용프로그램등을 사용해서 만들 수 있다.</p> <table> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-b</td><td>빈 행의 번호는 출력하지 않는다.</td></tr> <tr> <td>-n</td><td>빈 행의 번호도 출력한다.</td></tr> <tr> <td>-s</td><td>중복되는 빈 행은 하나의 빈 행으로 출력한다.</td></tr> </table>	옵 션	설 명	-b	빈 행의 번호는 출력하지 않는다.	-n	빈 행의 번호도 출력한다.	-s	중복되는 빈 행은 하나의 빈 행으로 출력한다.
옵 션	설 명								
-b	빈 행의 번호는 출력하지 않는다.								
-n	빈 행의 번호도 출력한다.								
-s	중복되는 빈 행은 하나의 빈 행으로 출력한다.								

¹ 방향 재지정 (Redirection - 리다이렉션)

리다이렉션은 표준 입력이나 표준 출력을 꼭 기본적인 입출력 장치인 키보드나 터미널로 하지 않고 방향을 바꾸어 파일 또는 다른 장치로 출력이나 입력의 방향을 재 지정하는 것을 말한다. 표준 입력을 바꿀 때는 "<" 또는 "<<"을 사용하여 뒤에 붙은 파일이나 디바이스로 부터 입력을 받으며, 표준 출력을 바꿀 때는 ">" 나 ">>"을 사용하여 표준입력으로 부터의 입력사항을 뒤에 붙은 디바이스나 파일로 출력해준다. 따라서, "**cat > a.txt**" 명령의 경우는 표준입력(키보드)로 부터 입력을 받아 **a.txt**라는 파일로 출력을 하여 파일을 만들게 된다. 마지막으로 [Ctrl]-[D]키를 사용하여 파일의 마지막임을 알려주어야 한다.

"**cat > a.txt**" 와 "**cat >> a.txt**"의 차이점은 전자는 **a.txt**파일이 존재할 경우 임의로 덮어쓰고 후자의 경우는 **a.txt**파일 내용다음에 다음에 추가된다.

cp	파일을 복사한다	
	옵션	설 명
	-f	덮어 씌울지 묻지않고 강제로 복사한다.
	-d	링크정보도 함께 복사한다.
	-p	작성시간, 퍼미션등의 정보도 함께 복사한다.
	-r	서브 디렉토리도 모두 복사한다.
	-a	원본 파일의 속성, 링크 정보들을 그대로 유지하면서 복사한다. (dpR 과 동일)
	-u	소스 파일이 복사될 파일보다 새것이거나 복사될 파일이 없을때만 복사한다
rm	-v	작업진행내용 출력한다..
	파일 또는 디렉토리를 삭제한다. 특히 디렉토리 삭제의 경우 디렉토리가 비워져 있어야 지워지는 rmdir보다 [rm -r]이 사용하기 쉽다.	
	옵션	설 명
	-f	지울지 여부의 메시지를 보여주지 않고 강제로 삭제한다
	-r	디렉토리의 내용을 삭제한다.
file	-i	삭제할 파일들에 대해 각각 확인메시지를 출력하여 확인후 삭제한다.
	-v	작업진행내용을 출력한다..
file	파일의 종류를 확인하는 명령어로 우리는 아래와 같이 이 명령이 ARM용 인지를 확인할 때 사용된다. 하지만 항상 정확한 정보를 제공하는 것은 아니므로 무조건 신뢰하지는 말자.	
	<pre>[root@Developers arm_example]# file ./hello ./hello: ELF 32-bit LSB executable, ARM, version 1, dynamically linked (uses shared libs), not stripped</pre>	
ln	링크 파일 ¹ 을 생성하는 명령으로, 이 링크파일은 하나의 파일을 두개 이상의 이름으로 접근이 가능하게 한다.	
	옵션	설 명
	-s	심볼릭 링크를 생성한다.
dd	파일을 읽어 변환하여 복사한다.	
	부팅 디스크 만들 때 dd if=[파일] of=[파일] bs=[byte]의 형식으로 만든다.	
dd	dd if =/boot/vmlinuz-2.2.14 of=/dev/fd0 bs=1k	

2.2. 파일 퍼미션(Permission) 관련

LINUX에서는 멀티유저가 가능하기 때문에 각 파일에 대해 각 계정에 대해 접근 권한을 주어져야 한다. 예를 들어 시스템파일을 루트가 아닌 다른 계정의 사용자가 임의로 삭제하거나 한다면 그 시스템은 심각한 오류를 겪게 되므로 시스템파일

¹ ※ 링크 (Link) 종류

① 하드링크

파일의 모든 속성을 그대로 유지하는 복사본과 같고, 일반파일에서만 만들 수 있다.

② 심볼릭링크

단지 원본 파일이나 디렉토리를 가리키는 방향에 불과한 것으로 단축아이콘과 유사한 개념으로 파일에 대해 모두 만들 수 있다.

의 삭제권한은 루트계정에만 준다.

여기서는 이런 권한에 해당하는 파일속성, 읽기/쓰기/실행권한, 소유권 등에 대한 내용을 담는 퍼미션(Permission)에 대해 다루도록 한다.

권한은 소유자(Owner), 그룹(Group), 다른사람들 (Others) 의 3종류로 나누어 지며 아래에서 이것들에 대한 설명을 예제와 함께 할 것이다.

파일정보의 각 상세설명 ("ls -al" 명령으로 볼 수 있다.)

```
[root@Developers tmp]# ls -al
total 12
drwxr-xr-x  2 root  root    4096 Oct 25 16:24 .
drwxr-x--- 22 root  root    4096 Oct 25 13:20 ..
-rw-r--r--  1 root  root     21 Oct 25 16:24 mytext.txt
```

위의 예에서 mytext.txt 파일의 각 필드들이 나타내는 의미를 알아본다.

필드

내용

이 필드는 파일의 종류와 접근 권한 표시하며 아래와 같다.

	종류	User				Group			Others		
비트	1	2	3	4	5	6	7	8	9	10	
내용	종류	r	w	x	r	w	x	r	w	x	
예제	-	r	w	-	r	-	-	r	-	-	

① 첫 번째 문자 ("-")

첫 번째 문자는 파일의 종류를 표시한다. 파일의 종류엔 여러 가지가 있으며 각각의미는 다음과 같다.

"-"	일반파일
"d"	디렉토리
"b"	블록형 장치파일 (디스크장치 등과 같이 블록 단위로 입출력이 이루어지는 디바이스)
"c"	문자형 장치 파일 (RS-232C 회선이나 테이프 디바이스와 같이 문자 단위로 입출력이 이루어지는 디바이스)
"l"	심볼릭 링크
"s"	소켓 (프로세스간 통신을 하기위한 기능)
"p"	파이프

② 나머지 문자들의 의미

위의 전체 구조에서 최상위 1비트를 뺀 나머지는 "r", "w", "x"의 권한으로 구성되며 각 파일들에 따른 의미는 아래와 같다.

일반파일	<div>'r': 읽기 가능 'w': 쓰기 가능 'x' : 실행가능</div> <div>'-': 해당권한을 주지 않음</div>
디렉토리	<div>'r': 읽기 가능 (ls 명령으로 볼수 있음을 의미)</div> <div>'w': 쓰기권한 디렉토리 내에 파일이나 하위 디렉토리를 생성할 수 있음과 디렉토리의 이름을 변경할 수 있음을 의미</div> <div>'x' : cd 명령으로 그 디렉토리내에 들어 갈 수 있음을 의미</div>

첫번째

	<p>특수파일</p> <p>하드디스크, 시디롬 드라이브, 테이프장치등 모든 장치들을 하나의 파일로서 다루며 이러한 디바이스 장치 파일들을 <code>/dev</code> 디렉토리 아래 있는 파일들을 말한다.</p> <p><code>mknod</code> 라는 명령을 사용해서 생성할 수 있는데 블록형(b type), 문자형(c type)의 특수파일은 루트 사용자만 실행 가능하다.</p>
두번째	(1) 는 링크된 숫자를 나타낸다. (소프트 링크와 하드링크 참조)
세번째	파일의 소유자 (파일을 생성한 사용자)를 나타낸다. (root)
네번째	(root) 는 파일의 소유자가 속한 그룹을 나타낸다
다섯번째	파일의 크기 (21)을 나타낸다
여섯번째	파일을 마지막으로 수정한 날짜 (21 Oct 25 ..)를 나타낸다
일곱번째	파일명(mytext.txt)를 나타낸다

즉 위의 예제에 나온 "**mytext.txt**" 파일은 파일 소유자는 **root**계정이며, 이 소유자는 이 파일에 대하여 읽기 쓰기를 할 수 있고 사용자가 속한 그룹내의 사용자와 그룹 밖의 소유자들이 읽기 만을 할 수 있는 파일임을 의미한다. 이상에서 퍼미션 정보의 형식에 대해 알아 보았으며 아래에서는 이 퍼미션을 다루는 명령들에 대해 보도록 한다.

명 령	내 용
chmod	<p>파일에 대한 속성(퍼미션¹)을 변경하는 것으로 각각 파일소유자(u), 파일 소유자그룹(g), 그밖의 다른 사용자(o), 모든 사용자(a)에 대한 파일 속성을 바꾸는데 사용하는 명령으로 파일 속성에는 읽기권한(r), 쓰기권한(w), 실행권한(x)가 있으며 권한 부여는 숫자로 사용하는 방법과 문자로 사용하는 방법이 있다. 문자로 사용하는 방법에서는, '+'는 권한을 부여할 때 사용하고, '-'는 주어진 권한을 없앨 때 사용한다. 아래와 같다.</p> <pre> /*소유자에 대해 실행권한을 준다*/ [sohnet@localhost Mytest]\$ chmod u+x mylink [sohnet@localhost Mytest]\$ ls -l total 4 -rwxrw-r-- 1 sohnet sohnet 274 Jan 31 13:16 mylink* /*소유자에 대해 실행권한을 없앤다*/ [sohnet@localhost Mytest]\$ chmod u-x mylink [sohnet@localhost Mytest]\$ ls -l; total 4 -rw-rw-r-- 1 sohnet sohnet 274 Jan 31 13:16 mylink [sohnet@localhost Mytest]\$ chmod u+x,g-w,o-w *</pre> <p>현재 디렉토리의 모든 파일에 대해 '소유자(u)' 실행권한 부여, '그룹(g)' 쓰기 권한 금지, '타인(o)' 쓰기 권한 금지.</p> <p>숫자로 표시하는 방법이 어렵다고 처음에는 느껴질 수 있지만 익숙해지면 더 편리하다. 각 허가설정의 숫자는 아래와 같으며 합계형식으로 사용하면 된다.</p> <div> <div>400</div> <div>소유자 읽기</div> </div>

	<table> <tr><td>200</td><td>소유자 쓰기</td></tr> <tr><td>100</td><td>소유자 실행</td></tr> <tr><td>40</td><td>소유그룹 읽기</td></tr> <tr><td>20</td><td>소유그룹 쓰기</td></tr> <tr><td>10</td><td>소유그룹 실행</td></tr> <tr><td>4</td><td>기타사용자 읽기</td></tr> <tr><td>2</td><td>기타사용자 쓰기</td></tr> <tr><td>1</td><td>기타사용자 실행</td></tr> </table>	200	소유자 쓰기	100	소유자 실행	40	소유그룹 읽기	20	소유그룹 쓰기	10	소유그룹 실행	4	기타사용자 읽기	2	기타사용자 쓰기	1	기타사용자 실행
200	소유자 쓰기																
100	소유자 실행																
40	소유그룹 읽기																
20	소유그룹 쓰기																
10	소유그룹 실행																
4	기타사용자 읽기																
2	기타사용자 쓰기																
1	기타사용자 실행																
	<pre>[sohnet@localhost Mytest]\$ chmod 664 mylink</pre> <p>-> 소유자 읽기/쓰기, 소유그룹 읽기/쓰기, 기타 사용자 읽기허용이 된다.</p> <pre>[sohnet@localhost Mytest]\$ chmod 777 myexe</pre> <p>-> 모든 그룹에 대해 읽기/쓰기/실행권한이 주어진다.</p>																
chown	<p>파일의 소유자를 다른 사람에게로 변경시킬 수 있다. '-R' 옵션을 써서 디렉토리내의 모든 파일들의 퍼미션을 변경시킬 수 있다. chgrp 와 동일 옵션이므로 아래를 참조하라.</p> <pre>[sohnet@localhost Mytest]\$ chown -R sohnet /home/document/</pre>																
chgrp	<p>파일의 그룹 소유권을 변경시킨다. 파일의 소유자나 슈퍼유저만이 파일의 그룹 소유권을 바꿀 수 있다.</p> <table> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-R</td><td>모든 하위 디렉토리와 파일들의 그룹 소유권을 변경한다.</td></tr> <tr> <td>-c</td><td>실제로 파일의 권한이 바뀐 파일만 출력한다..</td></tr> </table> <p>이상 설명한 옵션은 chown, chmod에 공통으로 적용된다.</p> <pre>[sohnet@localhost Mytest]\$ chgrp -R Study_A /home/document/*</pre> <p>/home/document/ 의 모든파일 및 서버 디렉토리까지 Study_A 그룹 소유권으로 바꾼다.</p>	옵 션	설 명	-R	모든 하위 디렉토리와 파일들의 그룹 소유권을 변경한다.	-c	실제로 파일의 권한이 바뀐 파일만 출력한다..										
옵 션	설 명																
-R	모든 하위 디렉토리와 파일들의 그룹 소유권을 변경한다.																
-c	실제로 파일의 권한이 바뀐 파일만 출력한다..																

2.3. 파일/문자열 검색 관련

이곳에서 다루게 되는 명령들은 파일 또는 문자열을 검색하는 기능을 가지며 무척이나 많이 사용하게 될 것이므로 여러 번 반복사용으로 익숙해지도록 해야 한다.

명 령	내 용
grep	<p>문자열의 특정한 패턴을 검색하는 명령어로 보통 파이프라인으로 다른 명령어들과 연동하여 사용되는 경우가 많다.</p> <pre>[sohnet@localhost Mytest]\$ grep "main" test.c</pre> <p>test.c 파일에서 "main"란 문자열이 있는 행을 출력한다..</p> <pre>[sohnet@localhost Mytest]\$ rpm -qa grep "ftp"</pre> <p>rpm -qa 결과에서 "ftp"라는 문자열이 있는 행을 출력한다..</p>
find	<p>찾고자 하는 파일과 디렉토리를 검색한다. 이 find 명령은 다양한 옵션을 가지고 있으며, 이 들 옵션들을 지정함으로써 파일을 검색하고, 이 검색된 파일들에 대하여 특별한 일을 수행하도록 할 수 있다. 여기서는 자주 사용되는 옵션 몇 개만 보도록 할 것이며, man 명령을 사용하여 매뉴얼지를 읽어 보도록 하자.</p>

Find 명령 사용의 기본 형식은 `find [디렉토리] [옵션]` 이다.

옵 션	설 명
<code>-name "파일명"</code>	파일이름에 의해 검색한다.
<code>-perm 모드</code>	퍼미션 모드에 의해 검색한다.
<code>-type 파일종류</code>	파일종류에 의해 검색한다. 파일종류 : b (블록 장치파일), c (문자 장치파일), d (디렉토리), f (일반파일), l (심볼릭 링크된 파일), p (파이프), s (소켓)
<code>user 사용자</code>	파일소유자 ID에 의해 검색한다.
<code>-exec 명령</code>	검색하여 나온 파일들에 대해서 추가명령을 실행하는 것으로 세미콜론까지 사이의 명령을 실행한다. <code>{}</code> : 검색된 현재파일이름으로 치환 <code>\</code> : 셀에 의해 해석되는 것을 방지하기 위함. <code>;</code> : 명령어의 마지막

```
[sohnet@localhost Mytest]$ find / -perm 4000 -print
퍼미션에 따른 검색으로 / 디렉토리에서(서브디렉토리 포함) Set user ID가 걸려있는 파일을 찾아 출력한다. -print는 생략 가능하다.
[sohnet@localhost Mytest]$ find / -name "*.html"
루트디렉토리(/)에서(서브디렉토리 포함) .html로 끝나는 파일을 찾는다.
[sohnet@localhost Mytest]$ find / (-name *.html -o -name *.txt) -exec rm {} \;
```

이 명령은 루트디렉토리(/)에서(서브디렉토리포함) 이름이 *.html 또는(-o 옵션) 이름이 *.txt인 파일들을 찾아 지운다.(-exec rm 옵션). 여기서 보면 -o 옵션은 OR를 -a 옵션은 AND로 사용된다.

2.4. 시스템 관련

여기서 보게 될 시스템 관리를 위한 명령들은 시스템상의 정보를 보거나 수정하기 위해 필요한 명령들이며, 어떤 명령들은 "루트권한"으로만 접근이 가능한 명령들도 있다.

명 령	내 용
clear	터미널의 창을 지운다. 도스의 "cls" 명령과 같다.
su	슈퍼유저로 변경이 가능한 명령이다. 이 슈퍼유저로 불리는 루트(root)의 계정은 모든 파일에 대해 파일에 설정된 읽기, 쓰기등의 권한에 관계없이 접근할 수 있다. <pre>[sohnet@Localhost /]\$ su -> 슈퍼유저로 변경 Password: [root@Localhost /# exit -> 일반유저로 로그인한 경우 일반유저로 변경 [root@Localhost /# su sohnet-> sohnet 계정으로 변경</pre>
halt, reboot, shutdown	<ul style="list-style-type: none"> halt 명령 시스템을 정지시킬 때 사용한다. reboot 명령

	<p>시스템을 끄거나 재부팅할 때 사용한다.</p> <ul style="list-style-type: none"> ▪ shutdown 명령 <p>시스템을 종료하거나 리부팅할 때 사용한다. [shutdown -r now]은 시스템을 지금 즉시 리부트 한다. 동일한 명령은 [reboot] 이다.</p> <pre>[root@localhost /sbin]# ./shutdown -h now</pre> <p>위 명령은 지금 즉시 종료하는 명령이다. 이 명령은 halt 명령과 같다.</p> <pre>[root@localhost /sbin]# ./ shutdown 02:30 "This system will be shut down at 02:00"</pre> <p>새벽 02시 00분에 종료하라는 명령으로 종료시간 5분전에는 로그인 금지되고 종료시간이 가까워지면 주기적으로 위의 메시지를 출력한다. 위 명령을 취소하기 위해서는 'shutdown -c' 명령을 주면 된다</p>																
adduser	<p>사용자 계정을 새로 등록하기 위해서는 adduser 명령을 사용하며 형식은 아래와 같다.</p> <p>adduser [-u uid [-o]] [-g group] [G roup,...] [-d home] [-s shell] [-c comment] [-m [-k template]] [-f inactive] [-e expire] [-p passwd] [-n] [-r] name</p> <table border="1"> <thead> <tr> <th>옵 션</th><th>설 명</th></tr> </thead> <tbody> <tr> <td>-u <uid></td><td>시스템에서 사용자를 구별하기 위해 사용하는 유저 ID로, /etc/passwd 파일을 참고해서 자동으로 붙여 지게 되는데, 특별한 번호를 부여하고자 할 경우에 사용한다.</td></tr> <tr> <td>-g <group></td><td>사용자가 소속될 그룹을 지정하며, 그룹을 지정하지 않으면 사용자 이름으로 새로운 그룹을 생성한다</td></tr> <tr> <td>-d <home></td><td>사용자의 홈 디렉토리 설정으로 지정하지 않으면 /home 디렉토리 아래 사용자의 이름으로 생성한다.</td></tr> <tr> <td>-s <shell></td><td>로그인 시 사용할 기본 셸을 지정한다. 지정하지 않으면 시스템상의 기본 셸로 지정된다.</td></tr> <tr> <td>-c <comment></td><td>/etc/passwd 파일 내에서 해당유저 항목의 comment 필드에 사용할 문자열을 지정한다.</td></tr> <tr> <td>-e <mm/dd/yy></td><td>사용자 계정의 유효기간을 설정함에 의해 임시계정으로 생성한다.</td></tr> <tr> <td>-p <passwd></td><td>패스워드를 설정한다 (passwd 명령으로 설정할 수도 있다)</td></tr> </tbody> </table> <pre>[root@localhost /root]# adduser -g guest -c 'Jang Seon Woong' sohnnet</pre>	옵 션	설 명	-u <uid>	시스템에서 사용자를 구별하기 위해 사용하는 유저 ID로, /etc/passwd 파일을 참고해서 자동으로 붙여 지게 되는데, 특별한 번호를 부여하고자 할 경우에 사용한다.	-g <group>	사용자가 소속될 그룹을 지정하며, 그룹을 지정하지 않으면 사용자 이름으로 새로운 그룹을 생성한다	-d <home>	사용자의 홈 디렉토리 설정으로 지정하지 않으면 /home 디렉토리 아래 사용자의 이름으로 생성한다.	-s <shell>	로그인 시 사용할 기본 셸을 지정한다. 지정하지 않으면 시스템상의 기본 셸로 지정된다.	-c <comment>	/etc/passwd 파일 내에서 해당유저 항목의 comment 필드에 사용할 문자열을 지정한다.	-e <mm/dd/yy>	사용자 계정의 유효기간을 설정함에 의해 임시계정으로 생성한다.	-p <passwd>	패스워드를 설정한다 (passwd 명령으로 설정할 수도 있다)
옵 션	설 명																
-u <uid>	시스템에서 사용자를 구별하기 위해 사용하는 유저 ID로, /etc/passwd 파일을 참고해서 자동으로 붙여 지게 되는데, 특별한 번호를 부여하고자 할 경우에 사용한다.																
-g <group>	사용자가 소속될 그룹을 지정하며, 그룹을 지정하지 않으면 사용자 이름으로 새로운 그룹을 생성한다																
-d <home>	사용자의 홈 디렉토리 설정으로 지정하지 않으면 /home 디렉토리 아래 사용자의 이름으로 생성한다.																
-s <shell>	로그인 시 사용할 기본 셸을 지정한다. 지정하지 않으면 시스템상의 기본 셸로 지정된다.																
-c <comment>	/etc/passwd 파일 내에서 해당유저 항목의 comment 필드에 사용할 문자열을 지정한다.																
-e <mm/dd/yy>	사용자 계정의 유효기간을 설정함에 의해 임시계정으로 생성한다.																
-p <passwd>	패스워드를 설정한다 (passwd 명령으로 설정할 수도 있다)																
userdel	<p>사용자(계정)를 삭제한다.</p> <p>'userdel' 이 실행되면 /etc/passwd, /etc/shadow, /etc/group에 기록되어 있는 계정의 정보를 삭제하고 '-r' 옵션을 줄 경우 계정의 홈 디렉토리도 지워준다.</p> <pre>[root@localhost /root]# userdel sohnnet</pre> <p>sohnnet 계정을 삭제한다.</p>																
passwd	<p>패스워드 입력 명령으로 "passwd 계정이름" 으로 바꿀 수 있다.</p> <pre>[root@localhost /root]# passwd sohnnet</pre> <p>Changing password for user sohnnet New UNIX password: ****</p>																
who 또는 w	<p>현재 로그인한 사용자들을 보고자 할 때 사용한다.</p> <p>자신의 계정정보를 보고 싶을 때는 [whoami]를 자세한 정보를 보고 싶을</p>																

	<p>때는 [who am i]를 사용한다.</p> <table border="1"> <thead> <tr> <th>옵 션</th><th>설 명</th></tr> </thead> <tbody> <tr> <td>-l</td><td>로그인한 사용자의 호스트명을 출력한다..</td></tr> <tr> <td>-a</td><td>로그인한 모든 사용자에 대한 모든 정보를 출력한다..</td></tr> </tbody> </table>	옵 션	설 명	-l	로그인한 사용자의 호스트명을 출력한다..	-a	로그인한 모든 사용자에 대한 모든 정보를 출력한다..								
옵 션	설 명														
-l	로그인한 사용자의 호스트명을 출력한다..														
-a	로그인한 모든 사용자에 대한 모든 정보를 출력한다..														
uname	현재 시스템의 정보를 표시한다. (-a 옵션)														
df/du	<p>디스크장치(하드디스크/시디롬) 와 디렉토리의 용량을 보고자 할 때 사용한다.</p> <p>■ df 명령</p> <p>현재 디스크의 사용량과 남은 용량을 출력한다..</p> <table border="1"> <thead> <tr> <th>옵 션</th><th>설 명</th></tr> </thead> <tbody> <tr> <td>-h</td><td>1K, M, G 등으로 출력한다..</td></tr> <tr> <td>-a</td><td>0블록을 가진 파일시스템까지 출력한다..</td></tr> </tbody> </table> <pre>[root@localhost /root]# df Filesystem 1k-blocks Used Available Use% Mounted on /dev/hda5 2721480 1047912 1673568 39% / /dev/hda1 52376 37168 15208 71% /boot</pre> <pre>[root@localhost /root]# df -h Filesystem Size Used Avail Use% Mounted on /dev/hda5 2.6G 1.0G 1.5G 39% / /dev/hda1 51M 37M 14M 71% /boot</pre> <p>■ du 명령</p> <p>옵션으로 지정한 디렉토리의 용량을 표시한다.</p> <table border="1"> <thead> <tr> <th>옵 션</th><th>설 명</th></tr> </thead> <tbody> <tr> <td>-h</td><td>옵션을 설정하면 1K, M, G 등으로 출력한다..</td></tr> <tr> <td>-s</td><td>인수로 지정된 디렉토리의 합계만 출력한다..</td></tr> <tr> <td>--max-depth</td><td>인수로 지정된 디렉토리의 하부디렉토리 깊이를 뜻하는 것으로 이 깊이 까지의 디렉토리용량을 출력한다..</td></tr> </tbody> </table> <pre>[root@localhost /root]# du -h --max-depth=1 /etc more 417k /etc/X11 98k /etc/gtk 512 /etc/opt 8.5k /etc/ppp 4.5k /etc/rpm 61k /etc/ssh 65k /etc/mail 269k /etc/rc.d 25k /etc/sgml 30k /etc/skel</pre>	옵 션	설 명	-h	1K, M, G 등으로 출력한다..	-a	0블록을 가진 파일시스템까지 출력한다..	옵 션	설 명	-h	옵션을 설정하면 1K, M, G 등으로 출력한다..	-s	인수로 지정된 디렉토리의 합계만 출력한다..	--max-depth	인수로 지정된 디렉토리의 하부디렉토리 깊이를 뜻하는 것으로 이 깊이 까지의 디렉토리용량을 출력한다..
옵 션	설 명														
-h	1K, M, G 등으로 출력한다..														
-a	0블록을 가진 파일시스템까지 출력한다..														
옵 션	설 명														
-h	옵션을 설정하면 1K, M, G 등으로 출력한다..														
-s	인수로 지정된 디렉토리의 합계만 출력한다..														
--max-depth	인수로 지정된 디렉토리의 하부디렉토리 깊이를 뜻하는 것으로 이 깊이 까지의 디렉토리용량을 출력한다..														
free	스왑영역, 램 사용량과 남은 용량 표시한다. 아무런 옵션도 표시 하지 않을 경우는 KB 단위로 표시하고 -m 옵션을 사용하면 MB 단위로 표시한다.														
id, hostname	[id] 명령은 UserId를 보고자 할 경우, [hostname] 명령은 시스템에 지정된 호스트의 이름을 보고자 할 경우에 사용한다.														
which, whatis, whereis	<p>■ which <명령어></p> <p>명령어의 위치를 출력한다.</p>														

	<p>■ whereis <명령어> 명령어의 실행파일 경로 및 소스 매뉴얼 페이지 파일의 경로를 출력한다.</p> <p>■ whatis <명령어> 명령어의 기본 기능을 출력한다.</p> <pre>[sohnet@localhost /root]\$ whereis ls ls: /bin/ls /usr/share/man/man1/ls.1.gz [sohnet@localhost /root]\$ which ls alias ls='ls --color=tty' /bin/ls [sohnet@localhost /root]\$ whatis ls ls (1) - list directory contents</pre>
date	현재시간과 날짜를 출력한다.
cal	<p>달력을 출력한다. 아무런 인자를 주지 않으면, 현재 달을 출력한다.</p> <pre>[root@localhost lecture]# cal 7 1999 July 1999 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</pre>
ps, pstree	<p>[ps] 명령은 현재 실행중인 프로세스(작업, 프로그램)를 볼 때 사용되며 [pstree] 명령은 트리형태로 출력한다..</p> <pre>[sohnet@localhost sohnet]\$ ps -u USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND sohnet 31234 0.0 2.0 2208 1272 pts/0 S 11:48 0:00 -bash</pre> <p>User : 프로세스를 실행 시킨 소유자 계정 PID : 프로세서 ID %CPU : 마지막 분동안 프로세스가 사용한 CPU 시간 백분율 %MEM : 마지막 분동안 프로세스가 사용한 메모리의 백분율 VSZ : 프로세스의 자료와 스택크기 (KB) RSS : 프로세스에 의해 사용되는 실제 메모리 용량(KB) TTY : 프로세스 제어 터미널 (t3=/dev/tty2) STAT : 프로세스 상태 P : 수행가능/수행중 T : 일시정지 D : 디스크 입출력 대기 같은 인터럽트 할 수 없는 대기 상태 S : 20초 미만의 짧게 잠들 I : 20이상 길게 잠들 Z : 좀비(zombie) 프로세스 START : 수행시작된 시간 TIME : 지금까지 사용된 CPU의 시간(분,초) COMMAND : 명령어 이름</p>
kill	<p>해당 프로세스의 프로세스 ID(PID)를 사용하여 종료한다.</p> <pre>[sohnet@localhost sohnet]\$ ps -u</pre>

	<pre> USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND sohnet 31234 0.0 2.0 2208 1272 pts/0 S 11:48 0:00 -bash [sohnet@localhost sohnet]\$ kill 31234 </pre>
finger	<p>시스템 유저들에 대한 정보를 출력한다..</p> <pre> [root@Localhost root]# finger sohnet Login: sohnet Name: Jang Seon Woong Directory: /home/sohnet Shell: /bin/bash Last login Mon Jun 23 17:01 (KST) on pts/2 from 218.53.171.51 No mail. No Plan. </pre>
dmesg	<p>LINUX의 부팅 중에 나타나는 메시지를 표시하며 이 곳에 설정된 내용들은 부팅 중에 제대로 동작했는지에 대한 디버깅할 때 유용하게 사용된다.</p>

관련정보

■ 시스템 부팅관련 파일들

/etc/issue	부팅시 처음 표시되는 문자열들이 포함된다.
/etc/issue.net	텔넷을 통하여 이 시스템에 연결하는 사용자에게 보여주는 문자열이 포함된다.
/etc/inittab	부팅시 init 프로그램에 의해 읽히고 해석되는 파일이다.
/etc/rc.d/rc	부팅시 init 프로그램에 의해 inittab파일을 해석하여 실행되는 스크립트이다. 이 스크립트에서 rc.sysinit스�크립트, 런레벨에 따른 rc1.d, rc.2...등의 스크립트, rc.local 설정파일들을 실행한다. 자세한 내용은 커널소스 분석중 부팅에서 프라프트까지에서 보도록 할 것이다.

■ 시스템 관련 파일들

/etc/passwd	<p>패스워드는 /etc/passwd파일로 들어가며, 이 파일의 내용은 아래와 같다.</p> <pre> [root@localhost /root]# more /etc/passwd root:x:0:0:root:/root:/bin/bash bin:x:1:1:bin:/bin: daemon:x:2:2:daemon:/sbin: mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash sohnet:x:500:500:/home/sohnet:/bin/bash guest1:x:501:501:/home/guest1:/bin/bash </pre> <p>위의 구성을 보면 모두 7개 필드로 구성되어 있으며 사용자 계정 이름 : 패스워드 : User ID : Group ID : 사용자 전체이름 : 홈디렉토리 : 셸 : 패스워드: 암호화 된 패스워드가 들어가는데 shadow패스워드를 사용하는 경우 /etc/shadow 파일에 따로 저장하고 passwd파일에는 x로 표시한다. shadow파일은 root 계정만 볼 수 있다</p>
/etc/group	그룹이름과 사용자에게 대한 정보가 들어있다.
/etc/securetty	로그인가능한 tty 디바이스 이름들에 대한 정보를 가진다. 이 파일은 중요하며, 우리가 시리얼 포트로 로그인 하고자 할때 이곳에 해당 디바이스의 이름을 등록하여야 한다. 만약 등록하지 않는다면, 부팅이 끝난후에 로그인할 수 없을 것이다.

/etc/shells

시스템에서 사용할 수 있는 셸의 이름들을 포함한다.

2.5. 셸(Shell)관련

셸은 사용자와 커널간의 인터페이스 역할을 해주는 명령어 해독기(Command Interpreter)이다. 우리가 LINUX에 로그인하고 명령을 입력하는 등의 작업들은 셸이 입력을 받아 해독하여 해당 업무를 수행하게 된다.

셸에는 Bourne셸, Korn 셸, Tc셸, Bash 셸 등 여러가지가 있으나 여기서는 LINUX 표준인 Bash 셸에 대하여 살펴보도록 한다. 셸의 기능중에는 스크립트로 프로그래밍이 가능하여 이를 셸 스크립트라 부르며 셸 스크립트는 수동으로 해야 하는 번거로운 여러 작업들을 자동으로 할 수 있는 하나의 방법을 제시하므로 자세히 공부해볼 필요성이 있다. 여기서 이에 관해서는 다루지 않으나, 꼭 인터넷정보 혹은 도서를 통하여 알아두길 권한다.

명 령	내 용
alias, unalias	<p>[alias] 명령은 긴 명령어와 옵션들을 간단한 별명으로 지정하여 사용할 수 있게 하며, [unalias] 명령은 alias명령을 사용하여 붙인 별명을 해제할 때 사용한다.</p> <p>이 [alias]에 대한 사항은 일반적으로 자신의 계정으로 로그인하면서 실행하게 되는 스크립트인 계정 홈 디렉토리에 위치해 있는 .bashrc 파일(Bash 셸의 경우)에 저장하여 로그인 시 자동으로 설정하도록 한다.</p> <pre>alias j='jobs'</pre> <p>위의 예는 jobs라는 명령어의 별명으로 j를 주고 있으므로 j를 치면 jobs를 친 것과 동일한 효과가 나타난다. 아래는 .bashrc 파일에 쓰고자 하는 별명을 미리 정의해 주었다.</p> <pre>[sohnet@localhost sohnet]\$ cat >> .bashrc alias cls='clear' alias m='more' [Ctrl]-[D] [sohnet@localhost sohnet]\$</pre>
환경변수 목록보기/ 바꾸기	<p>셸에서 사용하는 변수에는 셸 변수와 환경변수가 있다.</p> <p>■ 셸 변수</p> <p>사용자가 지정하여 사용할 수 있으며, 다음과 같이 "="를 사용하여 설정하고 "\$"를 붙여 값을 사용할 수 있다. 또한, 각종 명령어들과 연동하여 응용할 수도 있다.</p> <pre>[root@Localhost root]# SOH=babo [root@Localhost root]# echo \$SOH babo [root@Localhost root]#</pre> <p>■ 환경변수</p> <p>시스템에서 필요한 최소의 변수들이며, 대표적인 것이 PATH변수이다. 이 PATH변수에는 실행할 명령어를 탐색할 경로들이 정의되어 있으며, 여기에 지정되지 않은 디렉토리에 있는 명령의 경우 직접 그 디렉토리로 가거나 상위 디렉토리부터 모든 경로를 지정하여 실행시켜야 한다.</p>

```
[root@Localhost root]# echo $PATH
/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:
[root@Localhost root]#
```

■ export 명령

셸변수로 지정한 변수들은 현재 프로그램에만 유효하며, 이 프로그램에서 시작한 다른 프로그램에게도 적용시키고 싶다면 **[export]**를 사용하여 환경 변수로 만들어 주어야 한다.

```
[root@Localhost root]# export SOH=babo
```

관련정보

■ BASH 셸의 설정파일

셸의 초기설정을 위한 파일들은 로그인하면서 실행하며 이들은 각 계정의 홈 디렉토리에 숨김파일로 존재한다. 이 파일들중 **[.bashrc]**에는 보통 **Alias**등에 관한 설정을 두고, **[.bash_profile]**파일에는 **PATH**등의 환경변수들에 대한 정의를 둔다.

■ 명령어 입력을 쉽게하는 팁

▪ Tab 자동완성 키

알아두면 편한 기능으로 긴 파일명을 칠 때 앞에 한 두자만 치고 **Tab**키를 누르면 나머지가 매치되는 부분을 자동으로 채준다.

▪ history 명령

최근 실행한 명령을 다시 실행한다.

![번호] 를 입력하면 그 번호의 실행명령과 같은 것이 실행된다.

```
[sohnet@localhost sohnet]$ history | more
```

```
....
```

```
148 source .bashrc
```

```
149 alias
```

```
150 m .bashrc
```

```
151 cat .bashrc
```

```
152 history | more
```

```
[sohnet@localhost sohnet]$ !152
```

위의 **!152**는 152번의 **history | more**을 실행한다.

2.6. 파일시스템 관련

LINUX에서는 minix, ext, ext2, xiafs, hpfs, fat, msdos, umsods, vfat, proc, nfs, iso9660, smb, ncp, ufs, sysv 등 다양한 파일 시스템을 지원하며 이중 **ext2**(LINUX포맷), **msdos**(DOS), **vfat**(윈도우포맷), **iso9660**(CDROM포맷)등이 많이 쓰인다.

이들 파일시스템을 사용하기 위하여 **[mount]** 명령을 사용하며 이 명령은 현재 파일시스템의 한 디렉토리로 불러들여 사용가능 하도록 한다.

명 령	내 용						
mount	특정 파일시스템을 사용할 수 있도록 현재 파일시스템 특정 디렉토리로 연결시키는 작업을 마운트라 하며 [mount] 명령으로 수행된다.						
	<table><tr><th>옵 션</th><th>설 명</th></tr><tr><td>-v</td><td>마운트 작업을 수행중 작업정보를 출력한다..</td></tr><tr><td>-w</td><td>마운트 될 파일시스템에 대해 읽기/쓰기 동작을 허용한</td></tr></table>	옵 션	설 명	-v	마운트 작업을 수행중 작업정보를 출력한다..	-w	마운트 될 파일시스템에 대해 읽기/쓰기 동작을 허용한
	옵 션	설 명					
	-v	마운트 작업을 수행중 작업정보를 출력한다..					
-w	마운트 될 파일시스템에 대해 읽기/쓰기 동작을 허용한						

	<p>다.</p> <table border="1"> <tr> <td>-r</td><td>마운트 될 파일시스템에 대해 읽기동작만 허용한다.</td></tr> <tr> <td>-t</td><td>마운트 될 파일시스템의 타입을 지정한다</td></tr> </table> <p>아무 옵션없이 [mount] 명령만 수행하면 현재 마운트되어 있는 파일시스템들의 정보를 모두 출력한다..</p> <p>[CD-ROM] [sohnet@localhost sohnet]\$ mount -v -t iso9660 /dev/cdrom /mnt/cdrom 마운트할 장치명 : /dev/cdrom 이라 했는데 원래는 /dev/hdb(c/d/...)로 적어야 되지만 /dev/hdb를 /dev/cdrom으로 링크를 걸었기 때문에 /dev/cdrom이라고 써도 된다. 마운트할 디렉토리는 미리 만들어져 있어야 한다.</p> <p>[FLOPPY DISK] [sohnet@localhost sohnet]\$mount -v -t msdos /dev/fd0 /mnt/floppy</p> <p>[Hard disk의 다른 파티션을 마운트] /dev/hda1 파티션에 설치되어 있는 윈도우 98 시스템 마운트 [sohnet@localhost sohnet]\$mount -v -t vfat /dev/hda1 /mnt/win98</p>	-r	마운트 될 파일시스템에 대해 읽기동작만 허용한다.	-t	마운트 될 파일시스템의 타입을 지정한다
-r	마운트 될 파일시스템에 대해 읽기동작만 허용한다.				
-t	마운트 될 파일시스템의 타입을 지정한다				
umount	<p>마운트 한 파일시스템을 해제하는데 사용되는 명령이다. 현재 마운트된 디렉토리 안에 있거나 그 안의 프로그램이 실행중이라면 마운트된 디렉토리는 해제되지 않는다.</p> <p>[sohnet@localhost sohnet]\$umount /mnt/windows /mnt/windows에 마운트된 파일시스템을 언마운트 한다. [sohnet@localhost sohnet]\$umount -a 현재 마운트되어 있는 모든 파일시스템을 언마운트 시킨다.</p>				

2.7. 압축관련

인터넷이 발전하여 상호 정보교류가 활발한 요즈음 세상에서 파일크기를 줄이기 위한 압축기술은 여러 곳에서 사용된다. 실제로 LINUX상의 패키지나 파일들은 *.tar.gz, *.gz등의 대부분 압축파일들(물론 패키지는 RPM 패키지로 존재하는 것들도 많다)로 존재하며 인터넷에서 이런 종류의 파일들을 다운로드 받아 압축을 풀고 설치하고 하는 작업을 해야 하는 경우가 많다. 여기서는 이들 명령들에 대하여 살펴보고 하도록 하며 특히 가장 많이 쓰이는 **[tar]**명령에 대해서는 익숙해져야 한다.

명 령	내 용								
tar	<p>[tar] 명령은 압축은 하지 않고 파일들을 묶을 때 사용한다. 따라서, [tar] 명령독립적으로 사용하는 경우는 드물며 (-z) 옵션 사용으로 gzip을 통한 압축/해제를 수행한다. 이렇게 묶인 파일은 아카이브 파일이라 불린다.</p> <p>☞ 형식 tar [동작모드/옵션] [아카이브 파일명] [원본파일들]</p> <table border="1"> <thead> <tr> <th>동작모드</th><th>설 명</th></tr> </thead> <tbody> <tr> <td>-c</td><td>새로운 아카이브 파일을 생성한다</td></tr> <tr> <td>-x</td><td>기존에 만들어진 아카이브 파일을 푼다</td></tr> <tr> <td>-r</td><td>기존에 만들어진 아카이브 파일에 다른 파일들을 더 추가한다</td></tr> </tbody> </table>	동작모드	설 명	-c	새로운 아카이브 파일을 생성한다	-x	기존에 만들어진 아카이브 파일을 푼다	-r	기존에 만들어진 아카이브 파일에 다른 파일들을 더 추가한다
동작모드	설 명								
-c	새로운 아카이브 파일을 생성한다								
-x	기존에 만들어진 아카이브 파일을 푼다								
-r	기존에 만들어진 아카이브 파일에 다른 파일들을 더 추가한다								

	-v 작업진행내용을 화면에 출력한다..
split	<p>파일을 쪼갤때 사용되는 명령이다.</p> <p>split -b [size] [File] [suffix]</p> <p>[sohnet@localhost MyCTest]\$split -b 1440k X11R6.tar.gz x11r6</p> <p>3.5" 디스켓의 용량인 1.44MB 의 크기를 가진 x11r6aa,x11r6ab...로 쪼갬</p> <p>[sohnet@localhost MyCTest]\$cat x11r6* > x11r6.tar.gz</p> <p>쪼개진 파일을 하나로 뭉친다.</p>

2.8. RPM 패키지 관련

LINUX시스템에 새로운 프로그램을 설치하기 위하여서는 소스를 가져다 [configure] -> [make] -> [make install] 의 일련의 방식을 통하여 프로그램을 컴파일하여 설치해야 하는 어려움이 있었다. RPM(Redhat Package Management)은 이런 어려움을 해소하기 위하여 나온 프로그램설치/삭제등의 관리를 위한 프로그램이다. 여기서는 이 RPM의 사용법에 대하여 알아보도록 한다.

RPM 패키지의 이름구조는 아래와 같다.

<패키지이름>-<버전>-<패키지릴리즈 번호>.<아키텍처>.rpm

아래는 RPM 패키지들의 설치/제거/정보에 대한 RPM 명령의 사용법이다.

동 작	내 용												
패키지설치	<p>☞ 형식</p> <p>rpm - 옵션 [패키지이름]</p> <table> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-i</td><td>설치만 할 경우 사용하는 옵션으로 업그레이드를 할 경우에 이 옵션을 사용하면 충돌을 일으킬수 있으므로 되도록 사용하지 않는다.</td></tr> <tr> <td>-U</td><td>업그레이드를 할 때 사용하는 옵션으로, 높은 버전을 설치할 경우 이전 버전을 삭제하게 된다</td></tr> <tr> <td>-v</td><td>작업진행내용을 출력한다..</td></tr> <tr> <td>-h</td><td>설치 진행과정을 "#"로 출력한다..</td></tr> <tr> <td>--force</td><td>강제로 설치하라는 옵션으로, 이전 버전이 있거나 더 높은 버전이 있는 경우라도 무시하고 강제 설치한다.</td></tr> </table> <p>대개의 경우 -Uvh 옵션을 사용하여 설치한다. (rpm -Uvh [패키지이름])</p>	옵 션	설 명	-i	설치만 할 경우 사용하는 옵션으로 업그레이드를 할 경우에 이 옵션을 사용하면 충돌을 일으킬수 있으므로 되도록 사용하지 않는다.	-U	업그레이드를 할 때 사용하는 옵션으로, 높은 버전을 설치할 경우 이전 버전을 삭제하게 된다	-v	작업진행내용을 출력한다..	-h	설치 진행과정을 "#"로 출력한다..	--force	강제로 설치하라는 옵션으로, 이전 버전이 있거나 더 높은 버전이 있는 경우라도 무시하고 강제 설치한다.
옵 션	설 명												
-i	설치만 할 경우 사용하는 옵션으로 업그레이드를 할 경우에 이 옵션을 사용하면 충돌을 일으킬수 있으므로 되도록 사용하지 않는다.												
-U	업그레이드를 할 때 사용하는 옵션으로, 높은 버전을 설치할 경우 이전 버전을 삭제하게 된다												
-v	작업진행내용을 출력한다..												
-h	설치 진행과정을 "#"로 출력한다..												
--force	강제로 설치하라는 옵션으로, 이전 버전이 있거나 더 높은 버전이 있는 경우라도 무시하고 강제 설치한다.												
패키지제거	<p>☞ 형식</p> <p>rpm -e [패키지이름]</p> <table> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-e</td><td>패키지를 제거한다.</td></tr> </table>	옵 션	설 명	-e	패키지를 제거한다.								
옵 션	설 명												
-e	패키지를 제거한다.												
패키지정보	<p>패키지 찾기와 정보보기를 위하여서는 -q 옵션을 사용하며 부가적으로 아래와 같은 옵션들과 함께 사용된다.</p> <table> <tr> <th>부가옵션</th><th>설 명</th></tr> <tr> <td>-a</td><td>설치된 모든 패키지에 대해서 해당명령을 수행한다.</td></tr> </table>	부가옵션	설 명	-a	설치된 모든 패키지에 대해서 해당명령을 수행한다.								
부가옵션	설 명												
-a	설치된 모든 패키지에 대해서 해당명령을 수행한다.												

-i	패키지에 대한 간단한 정보를 출력한다..
-l	패키지가 포함하고 있는 파일 목록을 출력한다..
-d	해당 패키지와 포함된 문서 파일들을 출력한다..
-c	해당 패키지와 관련된 설정파일들을 출력한다..
-R	패키지가 의존하고 있는 파일 또는 패키지 목록을 출력한다..

```
[root@Localhost root]# rpm -qa |grep telnet
telnet-0.17-23hl
telnet-server-0.17-23hl
[root@Localhost root]# rpm -ql telnet
/usr/bin/telnet
/usr/share/man/man1/telnet.1.gz
[root@Localhost root]#
```

3. LINUX 네트워크 설정

LINUX X기반(GUI기반)의 컴퓨터에서는 설정/제어툴들이 그래픽형태로 존재하여 시스템에 대한 전반설정을 작업하기 쉽도록 되어있다. 하지만, 임베디드 LINUX 시스템에서는 이런 툴들을 사용하기 어려우므로, 네트워크 설정에 관련된 파일들, 명령어들을 알아봄으로써 IP, DNS등의 설정을 할 수 있도록 한다.

3.1. 네트워크 관련 설정파일들

아래는 네트워크에 관련된 파일들을 나타내며, 만약 파일을 직접 수정하여 설정이 끝났으면 시스템을 재부팅하거나 네트워크를 재활성시켜야 한다. 네트워크를 재활성시키는 방법은 아래와 같이 **network** 스크립트를 실행하면 되는데 이 **network**파일은 실행 스크립트로 아래와 같이 인터페이스의 활성화/비활성화를 담당하고 있다. 이 작업을 스크립트를 사용하지 않고 하려고 한다면, **ifconfig**명령과 **route**명령으로 해결할 수 있다. 실제로 임베디드시스템에서는 이렇게 사용하는 경우가 많다.

```
[root@Localhost etc]# /etc/rc.d/init.d/network restart
인터페이스 eth0 (을)를 종료함: [ 확인 ]
loopback 인터페이스를 종료함: [ 확인 ]
IPv4 패킷 forwarding을 종료하고 있습니다: [ 확인 ]
네트워크 매개 변수를 설정하고 있습니다: [ 확인 ]
loopback 인터페이스 활성화중 입니다: [ 확인 ]
eth0 인터페이스 활성화중 입니다: [ 확인 ]
[root@Localhost etc]#
```

```
[root@Localhost etc]# ifconfig eth0 down          비활성화
[root@Localhost etc]# ifconfig eth0 192.168.0.10 up      활성화
[root@Localhost etc]# route gw 192.168.0.1
```

설정파일	내 용
/etc/sysconfig/network	<p>네트워킹을 할 지 여부, 호스트네임, 도메인네임, 게이트웨이들을 설정한다.</p> <pre>[root@Develop etc]# cat sysconfig/network NETWORKING=yes FORWARD_IPV4=false HOSTNAME=Local.Develop DOMAINNAME=Develop GATEWAY=218.53.171.33 GATEWAYDEV=eth0</pre>
/etc/resolv.conf	<p>네임서버(Name Server)를 등록한다.</p> <pre>[root@Localhost etc]# cat /etc/resolv.conf nameserver 168.126.63.1 nameserver 218.150.160.133 search localhost -> 이부분은 자주 방문하는 사이트의 도메인 부분을 입력하면, 도메인 부분을 뺀 호스트 부분으로 쉽게 접속할 수 있도록 해준다.</pre>
/etc/hosts	<p>/etc/hosts 파일은 내부시스템에서 알려진 호스트들의 목록들을 설정해 놓고 DNS를 거치지 않고 바로 사용할 때 사용한다.</p> <p>127.0.0.1 은 자기 자신을 나타내는 호스트 이름이고 그 다음 웹서버를 사용하기 위하여 PC의 IP와 호스트 이름을 지정해 주어야 한다.</p> <pre>bash-2.05\$ cat /etc/hosts # Do not remove the following line, or various programs # that require network functionality will fail. 127.0.0.1 localhost.localdomain localhost 218.53.171.34 www.kelb.com</pre>

3.2. 네트워크 관련 명령군

네트워크를 설정하고 상태를 점검할 수 있는 명령들에 대해 보도록 하자.

명 령	내 용								
ifconfig	<p>네트워크 카드설정을 위한 명령이며, 부팅 시 자동으로 실행하길 원한다면 Bash의 경우 /etc/rc.d/rc.local 스크립트 파일에 써 놓으면 된다. 옵션없이 사용하면 현재 시스템에 설치된 인터페이스들에 대한 정보를 출력한다.</p> <p>이 명령을 실행하여 설치하였다면, 이에 대한 정보는 /etc/sysconfig/network-scripts/ifcfg-ethx 파일에 저장된다.</p> <p>☞ 형식</p> <pre>ifconfig [interface] ifconfig interface [aftype] options address ...</pre> <table border="1"> <thead> <tr> <th>옵 션</th><th>설 명</th></tr> </thead> <tbody> <tr> <td>interface</td><td>인터페이스 이름은 보통 ethx ((eth0, eth1..))로 시작된다.</td></tr> <tr> <td>up</td><td>인터페이스 카드를 활성화 시킬때 사용한다</td></tr> <tr> <td>down</td><td>인터페이스 카드를 비활성 시킬때 사용한다</td></tr> </tbody> </table>	옵 션	설 명	interface	인터페이스 이름은 보통 ethx ((eth0, eth1..))로 시작된다.	up	인터페이스 카드를 활성화 시킬때 사용한다	down	인터페이스 카드를 비활성 시킬때 사용한다
옵 션	설 명								
interface	인터페이스 이름은 보통 ethx ((eth0, eth1..))로 시작된다.								
up	인터페이스 카드를 활성화 시킬때 사용한다								
down	인터페이스 카드를 비활성 시킬때 사용한다								

	<table border="1"> <tr> <td>ip</td><td>할당된 IP를 설정한다</td></tr> <tr> <td>netmask</td><td>네트워크 마스크 값을 설정한다.</td></tr> </table> <pre> [root@Localhost etc]# ifconfig eth0 218.53.171.50 netmask 255.255.255.224 up [root@Localhost etc]# ifconfig eth0 Link encap:Ethernet HWaddr 00:40:2B:20:EB:3E inet addr:218.53.171.50 Bcast:218.53.171.63 Mask:255.255.255.224 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 RX packets:3174898 errors:0 dropped:0 overruns:0 frame:0 TX packets:1808758 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:100 RX bytes:738868129 (704.6 Mb) TX bytes:553386336 (527.7 Mb) Interrupt:9 Base address:0x2000 lo Link encap:Local Loopback inet addr:127.0.0.1 Mask:255.0.0.0 UP LOOPBACK RUNNING MTU:16436 Metric:1 RX packets:2085 errors:0 dropped:0 overruns:0 frame:0 TX packets:2085 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:0 RX bytes:159406 (155.6 Kb) TX bytes:159406 (155.6 Kb) [root@Localhost root]#ifconfig eth0 down [root@Localhost root]#ifconfig eth0 up [root@Localhost root]#ifconfig eth0 218.53.171.51 </pre>	ip	할당된 IP를 설정한다	netmask	네트워크 마스크 값을 설정한다.				
ip	할당된 IP를 설정한다								
netmask	네트워크 마스크 값을 설정한다.								
ping	<p>ICMP 프로토콜의 ECHO_REQUEST 데이터그램을 사용하는 명령으로 보통 상대방이 있는지 혹은 네트워크가 제대로 동작하는지에 대해서 가장 많이 사용되는 명령이다.</p> <table border="1"> <tr> <th>부가옵션</th><th>설 명</th></tr> <tr> <td>-i</td><td>몇 초 단위로 패킷을 보낼 것인지를 설정한다. (디폴트는 1초이다.)</td></tr> <tr> <td>-s</td><td>보낼 패킷 크기를 설정한다. (디폴트는 64바이트이며 최대값은 65,507바이트이다.)</td></tr> <tr> <td>-t</td><td>호스트가 살아있는 동안 계속하여 실행한다.</td></tr> </table>	부가옵션	설 명	-i	몇 초 단위로 패킷을 보낼 것인지를 설정한다. (디폴트는 1초이다.)	-s	보낼 패킷 크기를 설정한다. (디폴트는 64바이트이며 최대값은 65,507바이트이다.)	-t	호스트가 살아있는 동안 계속하여 실행한다.
부가옵션	설 명								
-i	몇 초 단위로 패킷을 보낼 것인지를 설정한다. (디폴트는 1초이다.)								
-s	보낼 패킷 크기를 설정한다. (디폴트는 64바이트이며 최대값은 65,507바이트이다.)								
-t	호스트가 살아있는 동안 계속하여 실행한다.								
netstat	<p>네트워크의 상태를 알아보기 위해 사용한다.</p> <table border="1"> <tr> <th>옵 션</th><th>설 명</th></tr> <tr> <td>-r, -route</td><td>라우팅 테이블을 출력한다..</td></tr> <tr> <td>-i, -interfaces</td><td>인터페이스의 테이블들을 출력한다..</td></tr> <tr> <td>-a</td><td>모든 정보를 출력한다..</td></tr> </table> <pre> [root@Localhost root]# netstat -i Kernel Interface table Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg eth0 1500 0 3216849 0 0 0 1817382 0 0 0 BMRU lo 16436 0 2093 0 0 0 2093 0 0 0 0 LRU </pre>	옵 션	설 명	-r, -route	라우팅 테이블을 출력한다..	-i, -interfaces	인터페이스의 테이블들을 출력한다..	-a	모든 정보를 출력한다..
옵 션	설 명								
-r, -route	라우팅 테이블을 출력한다..								
-i, -interfaces	인터페이스의 테이블들을 출력한다..								
-a	모든 정보를 출력한다..								

	<pre>[root@Localhost root]# netstat -r Kernel IP routing table Destination Gateway Genmask Flags MSS Window irtt Iface 218.53.171.32 * 255.255.255.224 U 0 0 0 eth0 127.0.0.0 * 255.0.0.0 U 0 0 0 lo default 218.53.171.33 0.0.0.0 UG 0 0 0 eth0</pre> <p>☞ U 는 up(activate), G는 gateway, H는 host route, D는 ICMP에서 얻은 route, GH가 같이 나타나면 중간 게이트웨이를 지난 Host를 뜻한다</p>
traceroute	<p>IP 패킷이 목적지에 닿을 때까지 지나가는 게이트웨이들을 차례로 출력한다..</p> <pre>[root@Localhost root]# traceroute yahoo.com traceroute to yahoo.com (66.218.71.198), 30 hops max, 38 byte packets 1 218.53.171.33 (218.53.171.33) 1.971 ms 1.955 ms 2.025 ms 2 211.210.35.121 (211.210.35.121) 9.282 ms 37.031 ms 8.960 ms</pre>
nslookup	<p>도메인을 가지고 IP를 알려고 할 때 혹은 그 반대의 경우에 쓰인다.</p> <pre>[root@Localhost root]# nslookup yahoo.com Note: nslookup is deprecated and may be removed from future releases. Consider using the `dig' or `host' programs instead. Run nslookup with the `-sil[ent]' option to prevent this message from appearing. Server: 168.126.63.1 Address: 168.126.63.1#53 Non-authoritative answer: Name: yahoo.com Address: 66.218.71.198 [root@Localhost root]# nslookup 66.218.71.198 Note: nslookup is deprecated and may be removed from future releases. Consider using the `dig' or `host' programs instead. Run nslookup with the `-sil[ent]' option to prevent this message from appearing. Server: 168.126.63.1 Address: 168.126.63.1#53 Non-authoritative answer: 198.71.218.66.in-addr.arpa name = w1.rc.vip.scd.yahoo.com. Authoritative answers can be found from: 71.218.66.in-addr.arpa nameserver = ns5.yahoo.com. 71.218.66.in-addr.arpa nameserver = ns4.yahoo.com. 71.218.66.in-addr.arpa nameserver = ns3.yahoo.com. 71.218.66.in-addr.arpa nameserver = ns2.yahoo.com. 71.218.66.in-addr.arpa nameserver = ns1.yahoo.com. ns5.yahoo.com internet address = 216.109.116.17 ns4.yahoo.com internet address = 63.250.206.138 ns3.yahoo.com internet address = 217.12.4.104</pre>

	<pre>ns2.yahoo.com internet address = 66.163.169.170 ns1.yahoo.com internet address = 66.218.71.63 [root@Localhost root]#</pre>
route	라우팅에 대한 정보를 설정하는 명령으로 게이트웨이를 설정한다.

관련정보

■ tcpdump

네트워크상의 패킷들을 스니핑(엿보기)하기 위한 프로그램으로 유명한 프로그램이다. 자신의 인터페이스에 어떤 패킷들이 돌아다니는지 알아볼 수 있다. 이 프로그램은 <ftp://ftp.ee.lbl.gov>에서 최신버전을 구할 수 있다.

3.3. 슈퍼데몬 (xinetd)

슈퍼데몬은 여러 개의 데몬을 관리하는 서버에서 동작하는 데몬을 말한다. 이 슈퍼데몬 xinetd는 `[/etc/xinetd.conf]`이 설정파일이 이며, `[/etc/services]` 파일에 설정된 포트번호에 대해서 클라이언트의 요청이 있을 때 필요한 데몬²을 실행시키게 된다.

■ 설정파일

```
[root@Localhost root]# cat /etc/xinetd.conf
defaults
{
    instances                = 60
    log_type                  = SYSLOG authpriv
    log_on_success             = HOST PID
    log_on_failure             = HOST
    cps                       = 25 30
}
includedir /etc/xinetd.d
```

마지막 라인에 `/etc/xinetd.d` 디렉토리를 참조하라고 나온 것을 볼 수 있다. 이 디렉토리에는 슈퍼데몬에서 관리하는 각종 데몬에 대한 설정파일들이 있으며, 이 설정파일로 슈퍼데몬에서 관리할 수 있도록 할 수 있다. 금지/허용에 해당하는 "disable" 항목의 값을 "no"로 하면 허용이 되어 실행된다.

예를 들어 telnet을 관리하는 telnet 설정파일을 보도록 하자.

```
[root@Localhost root]# cat /etc/xinetd.d/telnet
service telnet
disable          = yes      # 이 항목을 no로 하면 telnet서버를 사용할 수 있다.
flags            = REUSE
socket_type      = stream
wait            = no
user            = root      # 루트권한
server           = /usr/sbin/in.telnetd  # 프로그램실행경로
log_on_failure   += USERID
```

4. C 프로그램 만들기과 컴파일하기

이 장에서는 LINUX에서 C언어를 사용하여 간단한 프로그램을 만들고 gcc와 make의 간단한 사용을 통해 컴파일하는 방법에 간략히 살펴보도록 한다.

4.1. C 소스화일

여기서 C언어에 대한 구조나 문법같은 것에 대해서도 파일을 만드는 법에 대해서는 설명하지 않겠다. 바로 실습으로 들어간다.

아래와 같이 C언어 소스파일을 두개 만들도록 하자.

```
[root@Dci src]# cat > ver.c
unsigned int appver=1;
[Ctrl]-[D]키를 눌러 파일종료를 알린다.
[root@Dci src]# cat > test.c
#include <stdio.h>
extern unsigned int appver;
int main()
{
    printf("My Version Is %d!\n", appver);
    return 0;
}
[Ctrl]-[D]키를 눌러 파일종료를 알린다.
[root@Dci src]# ls
ver.c  test.c
```

4.2. Makefile

make 명령은 Makefile을 해석하여 컴파일을 도와주는 일종의 해석기이다. 일종의 배치파일 정도로 보면 되겠다. 이 make는 Makefile파일안의 내용을 기준으로 의존성검사, 종속성검사를 자동으로 하여 준다. 의존성검사는 하나의 파일이 생성되기 위하여 어떤 파일들이 필요한가를 검사하여 필요한 파일을 먼저 생성시키며, 종속성검사는 소스 중 컴파일 이후 최종 수정되어진 파일만을 새로 컴파일 시켜주는 작업을 한다. 앞으로 이 make명령에서 사용되는 Makefile파일은 수도 없이 접하게 될 것이므로 여기서 대략적인 구조를 살펴본다.

위에서 만든 C 파일들을 컴파일하여 최종 "myver" 실행파일을 만들기 위하여 아래와 같은 Makefile을 만들도록 하자.

```
1 # Makefile
2 CC= gcc
3 CFLAGS = -Wall -O2
4 SRCS=ver.c test.c
```

```

5 OBJS=ver.o test.o
6
7 myver : $(OBJS)
8         $(CC) $(OBJS) -o $@
9
10 ver.o : ver.c
11         $(CC) $(CFLAGS) -c ver.c
12
13 test.o : test.c
14         $(CC) $(CFLAGS) -c test.c
15
16 clean :
17         rm -f *.o
18         rm -f myver

```

위의 **Makefile**에서 앞의 숫자는 라인을 표시하기 위함이며, 실제로는 없다. 우선 형식상에서 맨 처음 칸에는 빈칸이 있으면 안된다. 빈칸이 있는 경우는 **8,11,14,17**과 같이 해당레이블의 동작 시 수행되어야 할 내용들이며, 이 빈곳에는 탭(**TAB**)이 들어가야 한다. 그럼 내용을 보도록 하자.

■ 1열

"#"으로 시작되는 것은 주석문이다. 해석에 아무런 영향을 주지 않는다.

■ 2열

CC는 환경변수를 선언하는 것으로 컴파일러를 **gcc**로 주고 있다. 환경변수의 이름은 관계없지만, 컴파일러의 경우 **CC**로 주는 것이 관행이다. 환경변수 안의 값은 "\$"를 붙여서 읽는다. 즉, \$(**CC**)는 **gcc**를 나타낸다.

■ 3열

CFLAGS 환경변수를 선언한다. 이는 컴파일러에 대한 플래그를 설정한다. 우선 **-Wall**은 경고메시지는 무시하라는 뜻이며, **-O2**는 최적화 옵션이다.

■ 4열, 5열

소스들의 이름과 생성될 오브젝트들의 이름을 환경변수로써 나열하였다 이들 환경변수들은 직접 \$(**CC**)는 **gcc**로 \$(**SRC**)는 "**ver.c test.c**"와 같이 풀어서 넣어도 상관이 없지만, 업그레이드 또는 프로젝트관리의 효율성을 위하여 이와 같이 환경변수로 만들도록 한다.

■ 7열

결국 **make**를 하면 **myver**라는 실행파일이 생긴다. 여기서 주목할 것은 "**myver:**"레이블 옆에 \$(**OBJS**)라고 해놓은 부분이다. 이는 종속성과 의존성부분으로 우선 의존성에 의해 **myver**가 생성되려면 \$(**OBJS**)라 되어 있는 부분 **ver.o** 와 **test.o**파일이 먼저 생성되어 있어야 함을 나타낸다. 따라서, 이 부분을 수행전에 **ver.o**를 생성하기 위한 **ver.o:**레이블과 **test.o**를 생성하기 위한 **test.o:**레이블부분이 먼저 실행된다. 종속성에 의한것은 **ver.o**나 **test.o**중 하나의 내용이

라도 변했다면 다시 생성시킬 것이다.

이들이 모두 생성되고 난 후의 최종 실행파일인 **myver** 실행파일은 `$(CC) $(OBJS) -o $@` 명령에 의해 생성된다. 여기서 **\$@**는 레이블의 이름 즉 **myver**와 같다.

■ 10열

ver.o : 레이블에서는 **ver.o**를 생성시키기 위한 부분으로 이전에 **ver.o**파일이 생성된 것이 있었다면 최후에 생성된 **ver.o**파일이후에 **ver.c**가 변하였는지를 살핀다. 만약 변하였다면 **ver.o**를 다시 생성시킨다. 당연히 **ver.o**가 없다면 그냥 생성시킨다.

`$(CC) $(CFLAGS) -c ver.c` 를 수행한다.

■ 13열

10열의 내용과 같다.

■ 16열

make clean 명령을 주었을 때 동작하는 레이블이다. **clean**을 인자로 주어 **clean** 레이블이 동작하도록 만든다. 이는 생성되었던 파일들을 모두 지워준다.

4.3. 실행파일 생성

그러면 위의 **Makefile**을 사용하여 컴파일을 하여 보자. "**make**" 단독 명령으로 모든 소스루틴들을 컴파일할 수도 있고, "**make myver**"로 "**myver**:"레이블을 지정하여 수행시킬 수도 있다. 여기서는 "**make myver**"를 사용하였다. 아래 "**make clean**"의 수행도 역시 해보자.

```
[root@Dci test]# ls
Makefile test.c ver.c
[root@Dci test]# make myver
gcc -Wall -O2 -c ver.c
gcc -Wall -O2 -c test.c
gcc ver.o test.o -o myver
[root@Dci test]# ls
Makefile myver test.c test.o ver.c ver.o
[root@Dci test]# ./myver
My Version is 1!
[root@Dci test]# make clean
rm -f *.o
rm -f myver
[root@Dci test]# ls
Makefile test.c ver.c
[root@Dci test]#
```

5. 커널컴파일하기

임베디드리눅스를 활용하기 위하여 커널을 컴파일하는 것은 거의 필수라는 것은

모두 알고 있는 사실일 것이다. 하지만, 꼭 임베디드리눅스를 하지 않더라도 LINUX를 사용하다 보면 커널을 업그레이하기 위하여 또는 커널을 최적화하기 위하여 커널을 컴파일해야 하는 경우가 있다. 여기서는 LINUX 시스템환경의 컴퓨터(보통 i386을 사용할 것이다.)에 대한 커널을 컴파일 하기 위한 대략적인 방법을 설명하며, 커널 컴파일과정에 대한 자세한 내용을 원한다면 <http://www.kldp.org> 사이트에서 커널 컴파일에 대한 문서를 찾아보길 권한다.

5.1. 커널 컴파일 준비하기

우선 현재 자신의 시스템의 커널 버전을 알아야 한다. 이는 `[uname -a]`명령으로 확인할 수 있다.

```
[root@Developers linux]# uname -a
Linux Developers.Localhost 2.4.7-10 #1 Thu Sep 6 17:27:27 EDT 2001 i686 unknown
[root@Developers linux]#
```

최신 커널소스는 <ftp.kernel.org/pub/linux/kernel/>사이트에서 최신의 커널 소스를 받으면 된다.

관련정보

■ LINUX버전

LINUX 버전에 대하여 잠시 보도록 하자. LINUX 버전의 구성은 아래와 같다.

<주 버전>.<부 버전>.<패치번호>

<주 버전>	커널의 변화가 클때 변하는 번호이다
<부 버전>	이 부버전의 번호는 홀수일때는 개발 버전, 즉 테스트버전을 나타내고, 짝수일 때는 안정된 버전을 뜻한다
<패치번호>	부버전 번호에서의 패치번호이다

5.2. 커널컴파일 명령순서

이하 아래의 모든 작업은 루트권한으로 커널소스의 최상위 디렉토리에서 이루어져야 한다.

make mrproper	이 명령은 이 전의 컴파일환경설정을 지워주기 위한 명령이다
----------------------	----------------------------------



make menuconfig	커널의 컴파일 환경설정을 위한 명령으로 이 외에 <code>make config</code> , <code>make xconfig</code> , <code>make menuconfig</code> 가 있으며 <code>menuconfig</code> 와 X환경 하에서 실행되는 <code>xconfig</code> 가 사용하기 쉽다.
------------------------	---



make dep	컴파일에 관련된 의존성 관계를 구축한다.
-----------------	------------------------



make clean	이 명령은 이 전의 컴파일중에 나온 오브젝트파일들을 없애주기 위한 명령이다.
-------------------	--

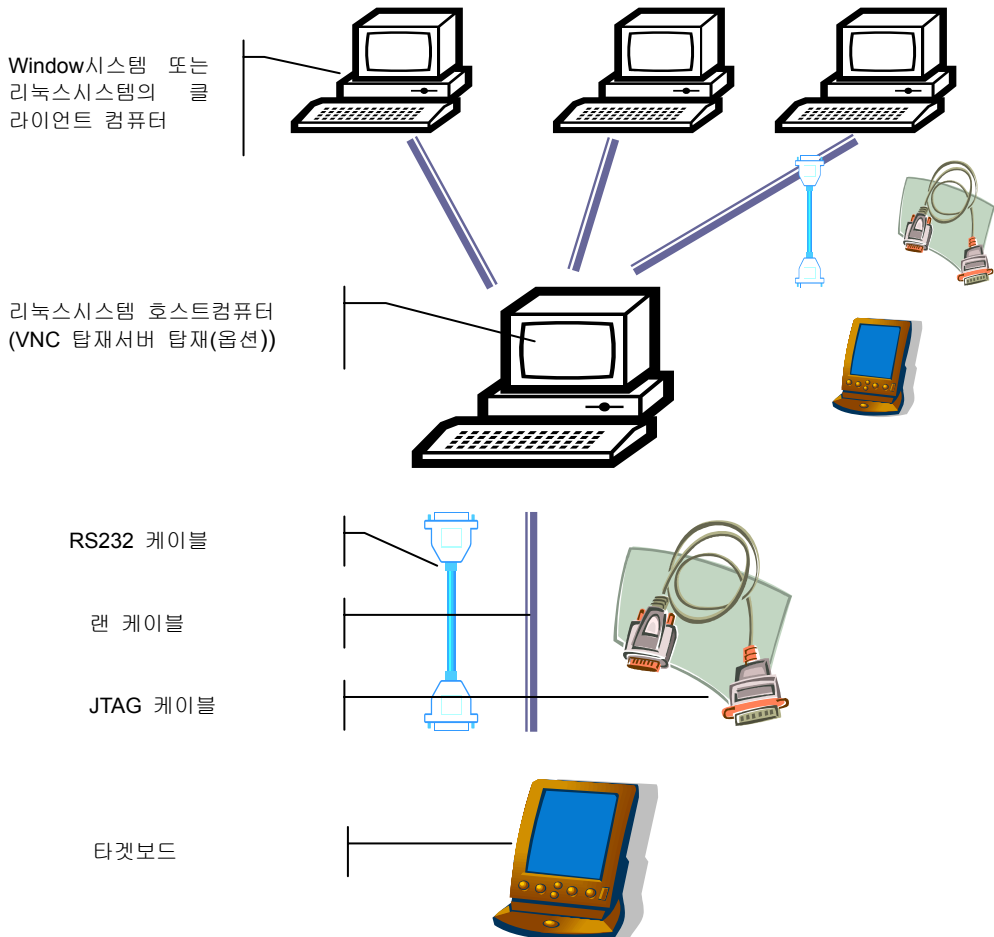


make bzImage	<p>컴파일을 하는 명령으로 커널이미지를 만들어 낸다. 이 커널 이미지는 소스최상위 디렉토리/arch/i386/boot/디렉토리에 bzImage가 생성된다.</p> <p>우리는 ARM용에서는 이 make zImage 명령을 사용할 것이다. 임베디드시스템의 경우 [make zImage]명령으로 생성된 zImage를 플래쉬롬에 써 넣는 작업으로 완료되지만, 호스트 컴퓨터의 경우 [make module] 명령 및 [make install] 명령을 수행해야 하며, Lilo 파일을 수정해야 한다.</p> <p>이 커널 컴파일 과정은 www.kldp.org 사이트에서 해당 문서를 찾아서 한번 정도는 해보도록 하자.</p>
---------------------	---

2

개발환경만들기

이 장을 시작하기 전에 LINUX가 설치된 호스트컴퓨터가 있고 이 호스트 컴퓨터에 호스트베이스의 프로그램을 개발할 수 있도록 기본 gcc등의 컴파일툴과 autoconf, make등의 호스트 개발툴이 설치되어 있어야 한다. 이것들은 보통 LINUX 배포판에 포함되어 있으므로 어렵지 않을 것이다.



실제적인 개발환경을 이를 LINUX시스템이 설치된 컴퓨터를 호스트컴퓨터라 하고, 클라이언트로서 LINUX 호스트컴퓨터에 접속하여 사용하게 될 윈도우베이스의 컴퓨터를 클라이언트컴퓨터, 그리고 실제적인 포팅타겟이 될 보드를 타겟보드로 부르도록 하겠다.

그림에서 클라이언트 컴퓨터는 굳이 필요없지만, 하나의 LINUX기반 호스트컴퓨터만을 구비하고 있는 경우 같은 환경에서 많은 사람이 함께 작업을 하고자 할 때는 유용하다. 이럴 경우 텔넷으로 서버에 접속하여 사용할 수 있겠지만, vi등 에디터의 사용이 어려우므로, 이후 장에서 나오는 Vnc라는 프로그램을 대안으로 제시하였다.

그러면 그림의 순서대로 각자 하는 일을 보기로 하자.

명 칭	역 할
클라이언트 컴퓨터	서버에 접속해 서버의 자원을 사용하여 작업을 할 수 있다. 특히 Window기반의 PC의 사용자일 경우 하나의 LINUX서버를 두는 것만으로 충분히 개발작업을 할 수 있다. 이렇게 쓸 경우 클라이언트 PC에서는 각자 다른 개발보드를 연결하여 디버깅 및 개발작업을 할 수 있다. Vnc를 사용할 경우 Vnc클라이언트가 설치되어 있어야 한다. 이에 대해서는 이후장에서 보도록 한다.
호스트컴퓨터 (LINUX기반)	포팅을 위한 실작업을 하기위한 모든 툴들이 설치되어 있다. 툴들의 설치에 대해서는 이후 장에서 본다.
JTAG	처음 부트로더를 플래쉬메모리에 써 넣기위해 사용한다. 부트로더 뿐 아니라 커널 및 램디스크등을 플래쉬에 써 넣을 수도 있지만, 속도가 느리므로 부트로더 이후에는 랜통신을 사용하는 것이 일반적이다. 물론, 하드웨어 디버깅을 위해 사용할 수 도 있다.
RS232	보드의 부팅메시지, 디버깅정보등을 볼 수 있으며, 로그인을 이 콘솔을 통하여 한다. LINUX서버에 연결되는 경우 PC의 COM포트에 연결하여 Minicom의 소프트웨어를 통하여 통신하고, Window 클라이언트 PC에 연결하는 경우 하이퍼터미널등의 소프트웨어를 이용하여 RS232통신을 한다.

랜	부트로더설치후 부트로더에서 지원한다면, TFTP를 통하여 커널, 램디스크등의 이미지를 다운로드 받고 플래쉬에 써 넣는다. 또한, 개발과정중 NFS를 통하여 서버의 디렉토리를 마운트하여 사용할 수 있으므로 디바이스드라이버등의 작업시 일일이 플래쉬에 써넣기 위하여 램디스크를 다시만드는 작업을 할 필요가 없으므로 무척 편리하게 사용할 수 있다.
타겟보드	LINUX 포팅을 위한 보드이다. RS232 포트가 장착되어 있어야 하며, 랜지원은 개발하려는 시스템에 필요가 없다면 없어도 되지만 개발당시에는 무척 불편하므로 있는 것이 좋다.

1. 시스템사용하기

1.1. 윈도우환경에서 작업하기(VNC)

윈도우환경에 익숙해져 있는 우리에게는 LINUX 환경이 개발을 위해서 적응하기 힘들 것이다. 이에 대한 대처방안으로 LINUX 환경의 개발컴퓨터와 윈도우 환경의 클라이언트 컴퓨터를 가지고 작업하는 방법을 택하도록 한다. 물론 telnet을 이용하여 접속하는 방법이 있기는 하지만, vi등의 에디터를 쓰는데 번거롭기 때문에 여기서 제안하는 VNC 프로그램을 사용하는 것이 좋을 듯 싶다.

이 프로그램을 사용하면 서버에 접속하여 서버의 모니터상의 환경과 똑 같은 윈도우를 볼 수 있으며, 서버의 모든 자원을 사용할 수 있다. 하지만, 서버와 클라이언트간의 파일전송은 하지 못하므로, FTP나 NFS를 사용하도록 한다.

여기서는 VNC 대신 같은 기능의 TightVnc를 사용하도록 하겠다. LINUX환경(호스트컴퓨터)-윈도우환경(클라이언트컴퓨터)에서 사용할 것이므로 <http://www.tightvnc.com> 에서 윈도우용과 LINUX용 두 가지 프로그램을 받도록 한다.

1.1. 윈도우 환경 VNC설치하기

[tightvnc-1.2.8-setup.exe](#) 파일을 다운로드 받는다. 이 파일을 설치하기만 하면 되고 [시작]-[프로그램]목록에 나타나고 Viewer를 뛰우면 서버에 접속할 수 있다. 접속방법은 이 후 과정에서 보게 될 것이다.

1.2. LINUX 환경 VNC설치하기

RPM 형태의 파일과 소스형태의 파일이 있는데 여기서는 소스형태의 파일인 [tightvnc-1.2.8_unixsrc.tar.gz](#)를 다운로드 받아 설치하는 과정을 보도록 하겠다. 이런 작업을 해 봄으로써 소스레벨의 어떤 응용소프트웨어 소스를 받아도 컴파일하고, 설치하는데 어려움을 느끼지 않을 것이다. 우선 압축을 풀고 나서 컴파일과 설치작업을 하기 전에 소스와 함께 따라오는 README 파일, INSTALL파일, ChangeLog

파일, WhatsNew파일 등을 읽어 보는 것이 중요하다.

1.2.1. 압축풀기

```
[root@Developers src]# pwd
/usr/src
[root@Developers src]# ls
tightvnc-1.2.8_unixsrc.tar.gz
[root@Developers src]#tar -xvzf tightvnc-1.2.8_unixsrc.tar.gz
.....
[root@Developers src]# ls
tightvnc-1.2.8_unixsrc.tar.gz  vnc_unixsrc
```

1.2.2. "xmkmf" 명령으로 Makefile을 만들고 "make World" 명령사용하여 컴파일한다.

이 명령은 TightVnc 소스트리의 최상위에서 실행되어야 한다.

```
[root@Developers vnc_unixsrc]# xmkmf
imake -DUseInstalled -I/usr/X11R6/lib/X11/config
[root@Developers vnc_unixsrc]# make World
```

1.2.3. XVnc 컴파일 하기

하위 디렉토리인 Xvnc로 이동하여 .configure 명령을 하고, make 명령으로 컴파일한다.

```
[root@Developers vnc_unixsrc]# cd Xvnc
[root@Developers Xvnc]# pwd
/usr/src/vnc_unixsrc/Xvnc
[root@Developers Xvnc]# ./configure
....
[root@Developers Xvnc]# make
.....
```

1.2.4. TightVnc 소스트리의 최상위로 이동하여 인스톨 하기

이제 컴파일이 모두 이루어 졌으므로, 이들 파일을 적당한 디렉토리에 설치해야 한다. 아래와 같이 실행파일들은 /usr/local/bin 디렉토리에 설치되고, 매뉴얼 파일들은 /usr/local/man 디렉토리에 설치된다.

```
[root@Developers Xvnc]# cd ..
[root@Developers vnc_unixsrc]# pwd
/usr/src/vnc_unixsrc
[root@Developers vnc_unixsrc]# ./vncinstall /usr/local/bin /usr/local/man
Copying Xvnc/programs/Xserver/Xvnc -> /usr/local/bin/Xvnc
Copying Xvnc/programs/Xserver/Xvnc.man -> /usr/local/man/man1/Xvnc.1
Copying vncviewer/vncviewer -> /usr/local/bin/vncviewer
Copying vncviewer/vncviewer.man -> /usr/local/man/man1/vncviewer.1
Copying vncpasswd/vncpasswd -> /usr/local/bin/vncpasswd
Copying vncpasswd/vncpasswd.man -> /usr/local/man/man1/vncpasswd.1
Copying vncconnect/vncconnect -> /usr/local/bin/vncconnect
Copying vncconnect/vncconnect.man -> /usr/local/man/man1/vncconnect.1
Copying vncserver -> /usr/local/bin/vncserver
Copying vncserver.man -> /usr/local/man/man1/vncserver.1
```

※ 위의 과정이 번거롭다고 생각되며 RPM 패키지로도 제공 되므로 tightvnc-

server-1.2.8-1.i386.rpm를 다운받아서 아래와 같이 설치한다.

```
[root@Developers sohnet]# rpm -Uvh tightvnc-server-1.2.8-1.i386.rpm
```

1.3. TightVnc 실습하기

설치가 모두 끝났으므로, 이제 서버(LINUX시스템)를 실행하고 클라이언트(윈도우 시스템)를 실행하여 접속해 보도록 하자.

1.3.1. vnc 패스워드 설정하기

클라이언트에서 서버로 접속하기 위한 패스워드를 설정한다.

```
[root@Developers vnc_unixsrc]# cd /usr/local/bin
[root@Developers bin]# ls
Xvnc      ftpwho    resize    vncconnect vncserver
ftpcount  hanterm   uxterm    vncpasswd  vncviewer
[root@Developers bin]# ./vncpasswd
Using password file /root/.vnc/passwd
Password:*****
Verify:*****
Would you like to enter a view-only password (y/n)? y
Password:*****
Verify:*****
[root@Developers bin]#
```

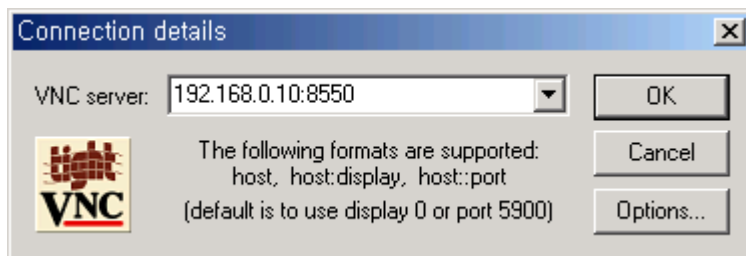
1.3.2. LINUX 환경의 시스템에서 서버로 실행하기

LINUX환경의 컴퓨터를 서버로 쓰기 위하여 /usr/local/bin디렉토리의 vncserver를 실행시킨다. 명령어 뒤의 숫자는 접속할 포트를 나타내는 것으로 이 숫자는 임의로 사용할 수 있다. 클라이언트에서는 여기서 설정한 포트번호로 접속되어야 한다.

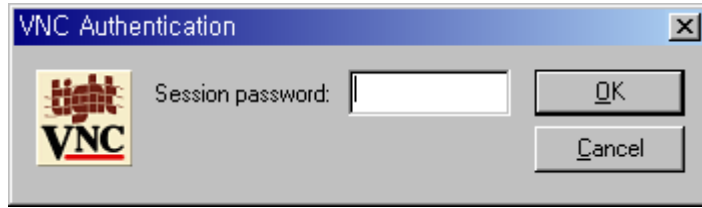
```
[root@Developers bin]# ./vncserver :8550
New 'X' desktop is Developers.Dci:8550
Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/Developers.Dci:8550
```

1.3.3. 윈도우 환경 시스템에서 서버 접속하기

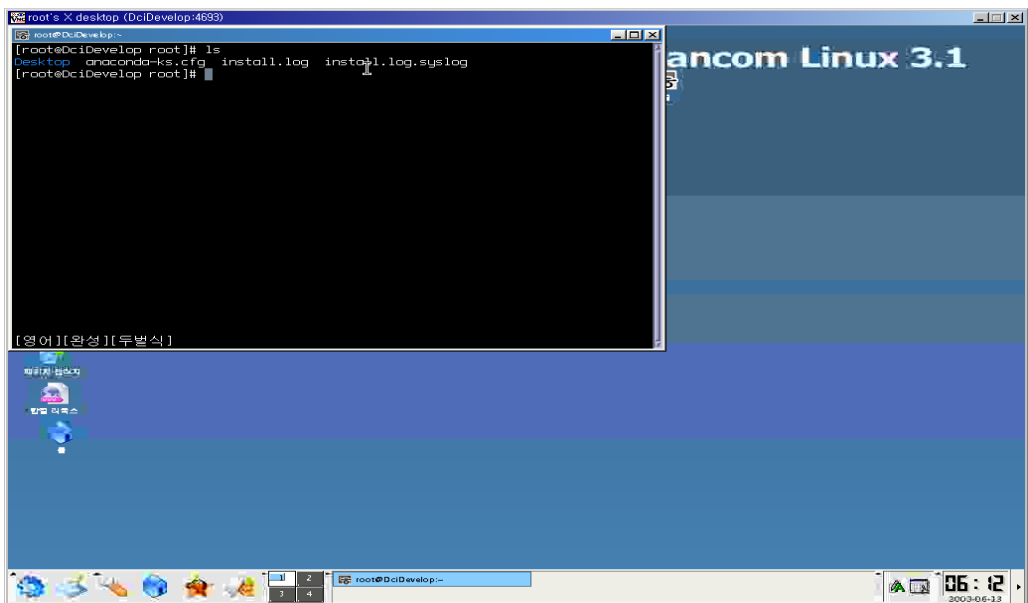
[시작]-[프로그램]-[TightVNC]-[TightVNC Viewer]프로그램을 실행시키도록 한다.



우리가 접속할 IP인 위의 LINUX 서버 IP를 적고, ‘:’ 문자 뒤에 위에서 설정한 포트번호를 적는다.



네트워크가 제대로 연결이 되었다면 위와 같이 패스워드를 묻는 창이 뜰 것이다. 여기에 위에서 설정한 패스워드를 지정한다. 연결이 되면 이제 모니터로 다음과 같이 리눅스 시스템상의 화면을 윈도우상에서 볼 수 있을 것이다. 멋지지 않은가. 프로그램의 빈공간에서 마우스를 누르면 메뉴가 나타난다.



※ 팁 : VNC 내부에서 텍스트를 블록을 잡고, 윈도우 하에서 [Ctrl-V]를 누르면 블록된 텍스트가 복사된다.

1.2. 시리얼 통신설정

개발보드와 통신을 하는데 처음에는 네트워크(랜) 기능의 통신기능이 활성화 되어 있지 않을 것이므로 가장 손쉽게 접근할 수 있는 시리얼통신을 통해 디버깅정보, 명령정보들을 받고 전달한다. 여기서는 이들 시리얼 통신을 할 수 있는 프로그램들에 대해 살펴보도록 하자.

1.2.1. 윈도우환경에서 하이퍼터미널을 이용하여 통신하기

윈도우 환경에서는 기본적으로 제공되는 시리얼통신 프로그램인 하이퍼터미널외에도 시리얼 통신을 제공하는 프로그램이 있다면 이를 사용하면 된다. 이 프로그램을 실행시켜 설정환경에서 보레이트, 데이터비트, Stop비트수, Start비트수등을 정해

주면 된다. 사용자들이 이 프로그램에 익숙하다고 생각되므로 여기에 대해서 더이상 설명하지 않는다.

1.2.2. LINUX시스템의 Minicom을 이용하여 통신하기

VNC를 사용하는 경우 작업이 용이하며, 여러명이 함께 작업을 할 수 있다는 장점을 갖지만, 2대 이상의 컴퓨터가 있어야 하는 단점이 있다. 한대의 컴퓨터를 가지고 작업을 할 때 타겟보드와 시리얼통신을 하기 위해 리눅스시스템 상에서 Minicom 프로그램을 사용한다. 이 프로그램의 설치 사용법에 대하여 간략히 보도록 한다.

■ 설치

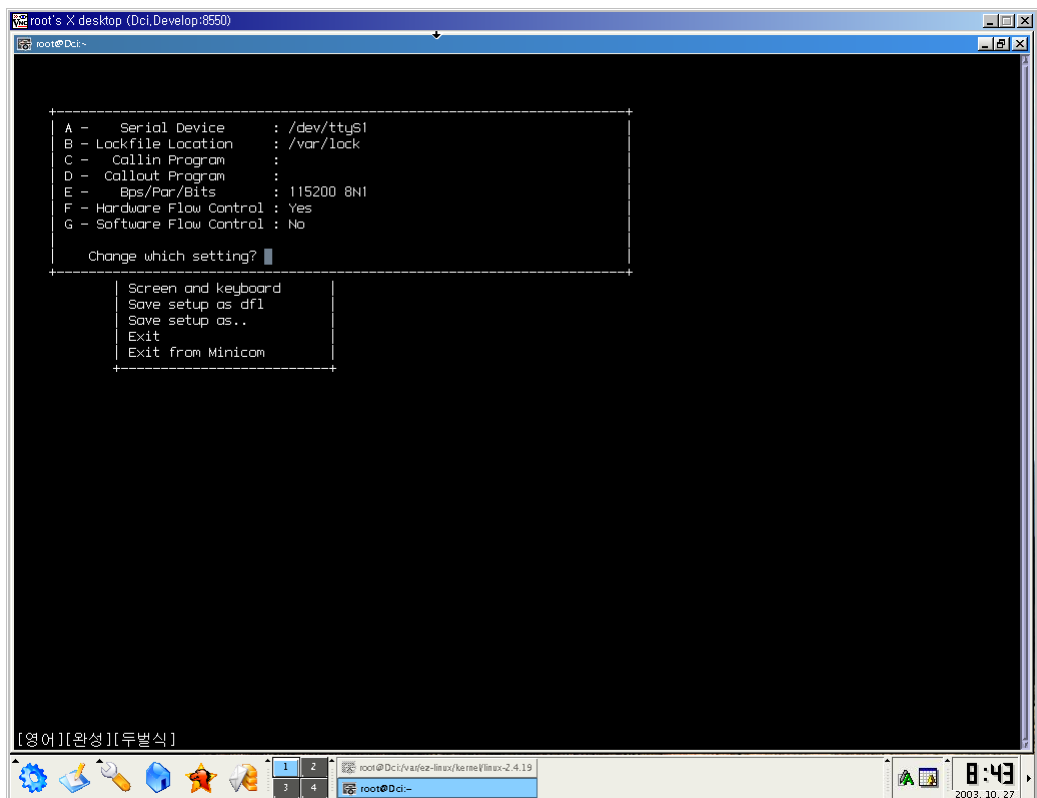
<http://rpmfind.net>에서 minicom에 대한 rpm 패키지를 받도록 하자. 필자의 경우 minicom-2.00.0-1cl.i386.rpm파일을 다운로드 받아 설치하였다.

```
[rpm -Uvh minicom-2.00.0-1cl.i386.rpm]
```

■ 실행

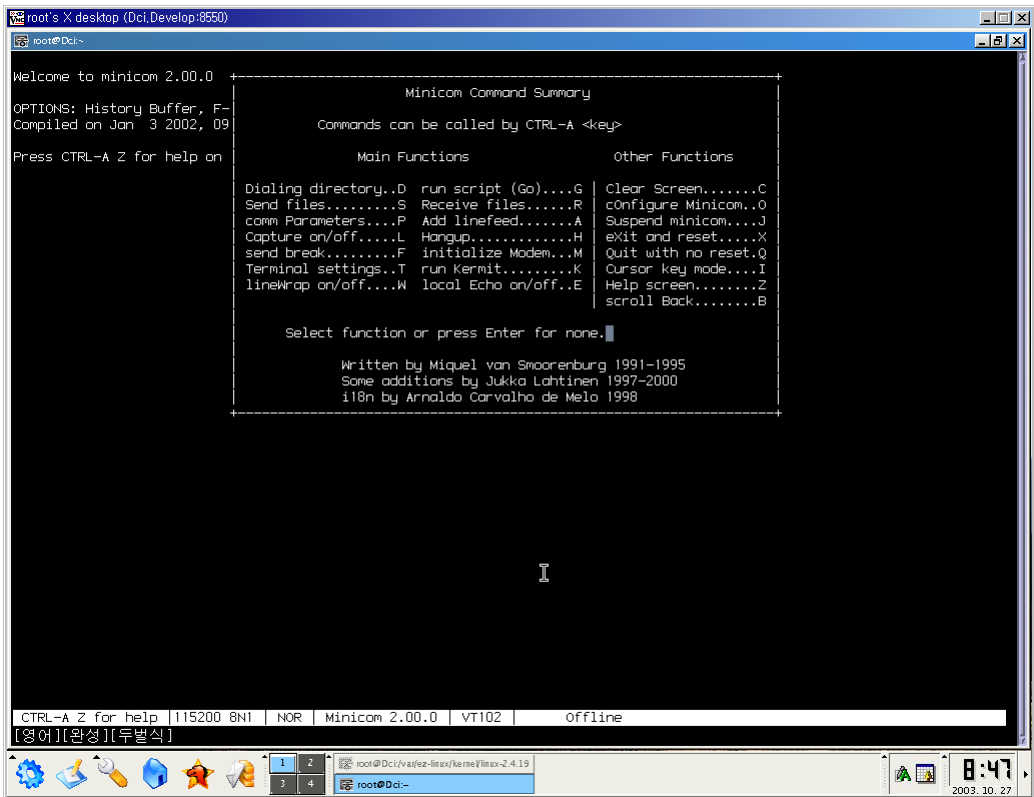
① 시리얼 포트의 설정의 하고 df1으로 설정하도록 한다.

- [minicom -s] 옵션으로 설정화면을 띄운다.
 - 시리얼 포트에 대해 보레이트, 데이터비트등의 전반 설정을 하도록 한다.
- [ESC]키는 현재 진행중인 메뉴로 부터 상위 메뉴로 간다.



- 설정을 끝낸다음 "Save setup as df1"항목을 선택하여 df1으로 저장한다.

- Exit항목을 선택하여 빠져나온다.
- [Ctrl-A] -> 'Z' 키를 차례로 눌러 메뉴가 뜨도록 하며 'X'를 눌러 빠져 나온다.
아래그림은 [Ctrl-A] -> 'Z' 키를 차례로 눌렀을 경우 뜨는 화면이다.



② mimicom의 사용

- [minicom]을 실행하면 df1의 파일에 설정한 대로 실행된다.
- 빠져나올 때는 [Ctrl-A] -> 'Z' 키를 차례로 눌러 메뉴가 뜨도록 하며 'X'를 눌러 빠져 나온다.

2. 크로스컴파일러 설치하기

LINUX 환경의 호스트 컴퓨터로 보통은 x86계열의 컴퓨터를 사용할 것이다. 이렇기 때문에 호스트 컴퓨터의 컴파일러를 가지고 작업을 한다면 결과 실행파일 또한 x86기반에서 동작할 수 있는 실행코드로 생성될 것이므로 이것이 실제적으로 동작할 타겟보드인 XScale 기반의 환경에서는 당연히 동작하지 않는다. 혹시라도, 타겟보드와 호스트컴퓨터의 CPU계열이 같다면 이 과정을 생략할 수 있다.

타겟보드에서 동작할 수 있는 실행파일을 생성해 내기 위해 호스트 컴퓨터(LINUX)에 XScale을 위한 컴파일러를 설치하여 이를 이용하여 커널 및 기타 프로그램을 컴파일 하여야 하며, 이 컴파일러를 크로스컴파일러라 한다.

크로스 컴파일러를 설치하기 위해 소스를 받아 어떠한 작업을 해야 하는지 알아보도록 하자. 이 설치작업은 리눅스를 포팅하기 위한 모든 작업의 기반이 될 것이므로 무척 중요하며, 아래의 내용은 필자의 컴퓨터에서 실행한 것임을 미리 알려둔다.

2.1. 소스레벨 패키지 다운로드 받기

Xscale용 크로스컴파일러를 만들기 위하여 해당하는 binutil파일, gcc파일, glibc파일이 필요하며, 패치로 Oerlikon-DevKit-XScalev2.tar.gz 파일을 적용할 것이다. 이 파일에는 binutil과 gcc의 해당버전에 대한 패치파일이 들어 있다.

📁 필요파일

종 류	다운로드 파일명
Xscale patch	ftp://ftp.arm.linux.org.uk/pub/linux/arm/toolchain/Oerlikon-DevKit-XScalev2.tar.gz
binutils	http://ftp.kernel.org/pub/linux/devel/binutils/binutils-2.13.90.0.16.tar.gz
gcc	ftp://ftp.gnu.org/gnu/gcc/gcc-3.2.1.tar.gz
glibc	ftp://ftp.gnu.org/gnu/glibc/glibc-2.3.1.tar.gz ftp://ftp.gnu.org/gnu/glibc/glibc-linuxthreads-2.3.1.tar.gz

2.1.1. Oerlikon-DevKit-XScalev2.tar.gz 압축풀기

```
[root@Dci 3.2.1]# pwd
/usr/src/Cross-Compiler/3.2.1
[root@Dci 3.2.1]# ls
Oerlikon-DevKit-XScalev2.tar.gz  binutils-2.13.90.0.16.tar.gz  glibc-2.3.1.tar.gz
gcc-3.2.1.tar.gz                glibc-linuxthreads-2.3.1.tar.gz
[root@Dci 3.2.1]#
[root@Dci 3.2.1]# tar -zxvf Oerlikon-DevKit-XScalev2.tar.gz
Oerlikon-DevKit-v2/
Oerlikon-DevKit-v2/INSTALL
Oerlikon-DevKit-v2/binutils-2.13.90.0.16-oerlikon.patch
Oerlikon-DevKit-v2/gcc-3.2.1-oerlikon.patch
Oerlikon-DevKit-v2/build-Oerlikon-DevKit-v2.sh
[root@Dci 3.2.1]# ls
Oerlikon-DevKit-XScalev2.tar.gz  binutils-2.13.90.0.16.tar.gz  glibc-2.3.1.tar.gz  Oerlikon-DevKit-v2
gcc-3.2.1.tar.gz                glibc-linuxthreads-2.3.1.tar.gz
[root@Dci 3.2.1]#
```

이하 아래 작업들은 위의 Oerlikon-DevKit-v2를 압축풀었을 때 나오는 스크립트인 build-Oerlikon-DevKit-v2.sh의 내용을 토대로 한다. 이 파일을 열어서 살펴보기 바란다.

2.2. binutil설치하기

2.2.1. 압축풀기 와 패치하기

```
[root@Dci 3.2.1]# tar -zxvf binutils-2.13.90.0.16.tar.gz
....
[root@Dci 3.2.1]# cd binutils-2.13.90.0.16
[root@Dci binutils-2.13.90.0.16]# pwd
/usr/src/Cross-Compiler/3.2.1/binutils-2.13.90.0.16
[root@Dci binutils-2.13.90.0.16]# patch -p1 < ../Oerlikon-DevKit-v2/binutils-2.13.90.0.16-oerlikon.patch
```

2.2.2. configure 실행하기

컴파일 하기 전에 컴파일을 위한 환경을 설정한다.

■ 설치 옵션

- ☞ --target 옵션 : 아키텍처
- ☞ --prefix 옵션 : 설치할 경로
- ☞ --with-cpu : 사용할 CPU

```
[root@Dci binutils-2.13.90.0.16]# ./configure --target=arm-linux --prefix=/usr/local/arm --with-cpu=xscale --nfp
...
```

여기서는 /usr/local/arm 디렉토리에 설치하고 있다.

관련정보

■ 에러가 난다면

CC와 OBJCOPY를 어디선가 export하여 쓴 적이 없는지 보라. 간혹 처음 부터 다시 설치한다고 이들 CC와 OBJCOPY를 다른값으로 (arm-linux-xxx)로 할당해 놓는 경우가 있다. 이럴 경우 아래와 같이 확인하고 처리하자. (Bash의 경우)

```
[root@Dci binutils-2.13.90.0.16]# echo $CC
arm-linux-gcc
[root@Dci binutils-2.13.90.0.16]# echo $OBJCOPY
arm-linux-objcopy
만약 그럴다면
[root@Dci binutils-2.13.90.0.16]# export CC=gcc
[root@Dci binutils-2.13.90.0.16]# export OBJCOPY=objcopy
```

2.2.3. make 실행하기

make 실행으로 컴파일하고, 에러가 발생하지 않는지 확인한다.

```
[root@Dci binutils-2.13.90.0.16]# make
...
```

2.2.4. make install 실행하기

make install 실행으로 우리가 설정한 설치 디렉토리(/usr/local/arm)에 설치하도록 한다. 이 명령 수행 후 해당 디렉토리에 디렉토리 및 파일들이 설치되어 있는가 확인하자.

```
[root@Dci binutils-2.13.90.0.16]# make install
...
[root@Dci binutils-2.13.90.0.16]# cd /usr/local/arm/
[root@Dci arm]# ls
```

```

arm-linux bin info lib man share
[root@Dci arm]# cd bin
[root@Dci bin]# ls
arm-linux-addr2line  arm-linux-c++filt  arm-linux-objcopy  arm-linux-readelf  arm-linux-strip
arm-linux-ar         arm-linux-ld       arm-linux-objdump  arm-linux-size     arm-linux-as
arm-linux-nm         arm-linux-ranlib   arm-linux-strings
[root@Dci bin]#

```

2.2.5. 경로추가하기 (Bash의 경우)

arm-linux-gcc 등의 명령을 사용하기 위하여 매번 절대경로를 사용하여 실행하기는 번거로우므로 "PATH" 환경변수에 이 경로를 추가하여 이곳에서도 명령어를 찾을 수 있도록 한다.

```

[root@Dci 3.2.1]# export PATH=$PATH:/usr/local/arm/bin
[root@Dci 3.2.1]# echo $PATH
/usr/local/sbin:/usr/sbin:/sbin:/usr/X11R6/bin:/opt/IBMJava2-13/bin:/home/sohnet/bin:/opt/IBMJava2-13/bin:/opt/IBMJava2-13/bin:/root/bin:/usr/local/arm/bin

```

2.3. LINUX 커널 소스 설치하기

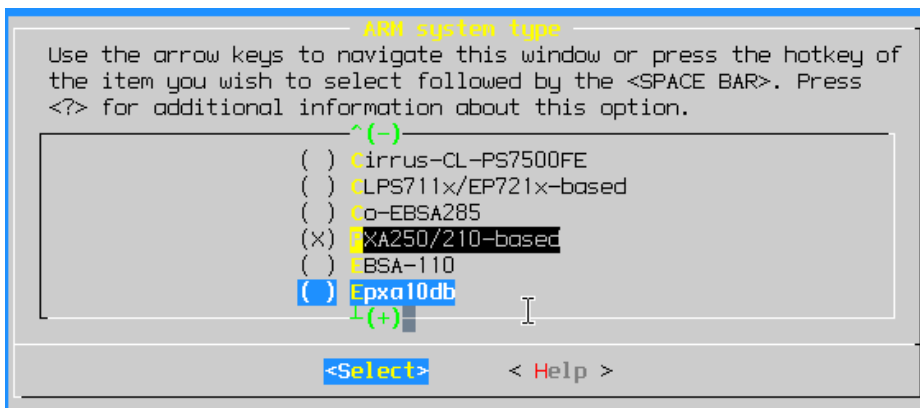
gcc를 컴파일 하기 전에 Linux 커널이 설치되어 있어야 한다. 이 커널 소스의 설치 는 부트로더의 [LINUX 소스 설치하기] 편을 참조하기 바란다.

저자는 이렇게 설치된 LINUX 소스의 디렉토리는 /usr/src/Kernel/linux에 설치하였다.

```

[root@Dci linux]# pwd
/usr/src/Kernel/linux
[root@Dci linux]# make menuconfig

```



위와 같이 커널소스 트리의 최상위 디렉토리에서 "make menuconfig" 명령을 실행 한 후 [system Type]을 "PX250/210-based"로 설정하고 빠져 나오도록 한다.

2.4. gcc c 컴파일러 설치하기

2.4.1.. gcc압축풀기와 패치하기

```
[root@Dci 3.2.1]# pwd
/usr/src/Cross-Compiler/3.2.1
[root@Dci 3.2.1]# ls
Oerlikon-DevKit-XScalev2.tar.gz  binutils-2.13.90.0.16
glibc-2.3.1.tar.gz
Oerlikon-DevKit-v2                binutils-2.13.90.0.16.tar.gz  gcc-3.2.1.tar.gz  glibc-
linuxthreads-2.3.1.tar.gz
[root@Dci 3.2.1]# tar -zxvf gcc-3.2.1.tar.gz
[root@Dci 3.2.1]# cd gcc-3.2.1
[root@Dci gcc-3.2.1]# patch -p1 < ../Oerlikon-DevKit-v2/gcc-3.2.1-oerlikon.patch
patching file gcc/reload1.c
```

2.4.2. 파일 수정하기

아래와 같이 gcc/config/arm/t-linux파일의 아래부분과 같이 수정해 준다.

```
[root@Dci gcc-3.2.1]# pwd
/usr/src/Cross-Compiler/3.2.1/gcc-3.2.1
[root@Dci gcc-3.2.1]# cd gcc/config/arm/
/usr/src/Cross-Compiler/3.2.1/gcc-3.2.1/gcc/config/arm
[root@Dci arm]# vi t-linux
  TARGET_LIBGCC2_CFLAGS = -fomit-frame-pointer -fPIC
-> TARGET_LIBGCC2_CFLAGS = -fomit-frame-pointer -fPIC
-Dinhibit_libc -D__gthr_posix_h 와 같이 추가한다.
  T_CFLAGS = -Dinhibit_libc -D__gthr_posix_h를 추가한다.
```

2.4.3. configure 명령 실행하기

■ 설치 옵션

- ☞ --target 옵션 : ARM용으로 arm-linux로 준다.
- ☞ --prefix옵션 : 설치할 경로를 설정한다.
- ☞ --with-headers옵션 : LINUX 커널소스의 include 디렉토리를 설정한다.

```
[root@Dci arm]# cd -
/usr/src/Cross-Compiler/3.2.1/gcc-3.2.1
[root@Dci gcc-3.2.1]# ./configure --target=arm-linux --prefix=$PREFIX $HOST --with-headers=$KERNEL/include --disable-shared --disable-threads --enable-languages="c" --nfp --with-cpu=xscale --without-fp --with-softfloat-support=internal
```

2.4.4. make 명령 실행하기

```
[root@Dci gcc-3.2.1]# make
```

컴파일 도중 에러가 발생하였다면, make clean 명령후 ./configure의 옵션을 살펴 보도록 하라.

2.4.5. make install실행하기

make install실행으로 우리가 설정한 설치 디렉토리(/usr/local/arm)에 설치하도록 한다. 이 명령 수행후 해당 디렉토리에 디렉토리 및 파일들이 설치되어 있는가 확인

하자.

```
[root@Dci gcc-3.2.1]# make install
...
[root@Dci gcc-3.2.1]# cd /usr/local/arm/bin
[root@Dci bin]# ls
arm-linux-addr2line  arm-linux-gcc      arm-linux-objcopy  arm-linux-strings
arm-linux-ar         arm-linux-gccbug   arm-linux-objdump  arm-linux-strip
arm-linux-as         arm-linux-gcov     arm-linux-ranlib
arm-linux-c++filt    arm-linux-ld       arm-linux-readelf
arm-linux-cpp        arm-linux-nm       arm-linux-size
[root@Dci bin]#
```

2.5. glibc 공유라이브러리 설치하기

2.5.1. glibc 압축 풀기

```
[root@Dci 3.2.1]# tar -zxvf glibc-2.3.1.tar.gz
```

2.5.2. glibc-linuxthreads 패치하기

```
[root@Dci 3.2.1]# cd glibc-2.3.1
[root@Dci glibc-2.3.1]# gzip -cd ../glibc-linuxthreads-2.3.1.tar.gz | tar xvf -
```

2.5.3. configure 명령 실행하기

위에서 설치한 gcc의 PATH를 설정해 주고 configure 명령을 수행한다. bash셸의 경우 아래와 같이 PATH에 /usr/local/arm/bin 디렉 토리를 추가하여 arm-linux-gcc를 찾을 수 있도록 해주어야 한다.

```
[root@Developers src]# export PATH=$PATH:/usr/local/arm/bin
[root@Developers glibc-2.3.1]# ./configure arm-linux --build=i686-pc-linux-gnu --with-headers=/usr/src/Kernel/linux/include --enable-add-ons --enable-shared --prefix=/usr/local/arm/arm-linux/lib --with-cpu=xscale --without-fp --enable-kernel=2.4.18
```

※ 위의 ./configure 명령시 나오는 메시지를 잘 확인하도록 하자. 만약 warning 이 있다면 에러가 날 수 있다. 이럴 경우 INSTALL 문서를 보고, 필요한 소프트웨어들의 버전을 확인하고 없다면 <http://rpmfind.net>에서 필요한 소프트웨어를 설치하도록 하자.

2.5.4. make, make install 명령 실행하기

```
[root@Dci glibc-2.3.1]# make
...
[root@Dci glibc-2.3.1]# make install
...
[root@Dci glibc-2.3.1]# cd /usr/local/arm
[root@Dci arm]# ls
arm-linux  bin  etc  include  info  lib  libexec  man  sbin  share
[root@Dci arm]# cd lib
[root@Dci lib]# ls
Mcr1.o          libdl-2.3.1.so      libnss_nis-2.3.1.so
crt1.o          libdl.a              libnss_nis.so
```


crti.o	libdl.so	libnss_nis.so.2
crtn.o	libdl.so.2	libnss_nisplus-2.3.1.so
.....		

2.5.5. 소프트 링크 잡아주기

위의 libc가 /usr/local/arm/lib에 잘 설치되었다면 이들 파일들을 /usr/local/arm/arm-linux/lib/ 디렉토리에 소프트 링크를 잡아주도록 한다.

```
[root@Dci glibc-2.3.1]# cd /usr/local/arm/arm-linux/lib/
[root@Dci lib]# ln -s /usr/local/arm/lib/* .
[root@Dci lib]# ls
Mcr1.o      libbsd-compat.a    libieee.a
.....
```

※ 이 작업을 하지 않으면 다음 gcc, c++ 설치하기에서 crt1.o파일을 찾을 수 없다는 에러 메시지를 보게 될 것이다.

2.6. gcc , c++ 설치하기

위에서 glibc를 설치하였으므로 c와 c++ 컴파일러를 다시 설치하여야 한다.

2.6.1. 이전 gcc 작업디렉토리 지우기

```
[root@Dci 3.2.1]# pwd
/usr/src/Cross-Compiler/3.2.1
[root@Dci 3.2.1]# ls
Oerlikon-DevKit-XScalev2.tar.gz  gcc-3.2.1.tar.gz
Oerlikon-DevKit-v2                glibc-2.3.1
binutils-2.13.90.0.16             glibc-2.3.1.tar.gz
binutils-2.13.90.0.16.tar.gz      glibc-linuxthreads-2.3.1.tar.gz
gcc-3.2.1
[root@Dci 3.2.1]# rm -rf gcc-3.2.1
```

2.6.2. gcc를 다시 설치한다.

```
[root@Dci 3.2.1]# pwd
/usr/src/Cross-Compiler/3.2.1
[root@Dci 3.2.1]# ls
Oerlikon-DevKit-XScalev2.tar.gz  binutils-2.13.90.0.16
glibc-2.3.1.tar.gz
Oerlikon-DevKit-v2                binutils-2.13.90.0.16.tar.gz  gcc-3.2.1.tar.gz  glibc-
linuxthreads-2.3.1.tar.gz
[root@Dci 3.2.1]# tar -zxvf gcc-3.2.1.tar.gz
[root@Dci 3.2.1]# cd gcc-3.2.1
[root@Dci gcc-3.2.1]# patch -p1 < ../Oerlikon-DevKit-v2/gcc-3.2.1-oerlikon.patch
patching file gcc/reload1.c
```

2.6.3. configure 명령 실행하기

■ 환경 옵션

--target : ARM용으로 arm-linux를 준다.

--prefix : 설치할 경로이다.

--with-headers : 위에서 설치한 glibc의 include디렉토리이다.

--with-libs : 위에서 서치한 glibc의 lib경로이다.

```
[root@Dci gcc-3.2.1]# pwd
/usr/src/Cross-Compiler/3.2.1/gcc-3.2.1
[root@Dci gcc-3.2.1]# ./configure --target=arm-linux --prefix=/usr/local/arm/ --with-headers=/usr/src/Kernel/linux/include --with-cpu=xscale --with-softfloat=internal --enable-languages=c,c++ --nfp
```

2.6.4. 파일수정하기

아래와 같이 netinetAddress.cc파일을 바꾸어 준다.

```
[root@Dci gcc-3.2.1]# vi libjava/java/net/natinetAddress.cc
▪ extern "C" int gethostname (char *name, int namelen);
-> extern "C" int gethostname (char *name, unsigned int namelen);
```

2.6.5. make 명령 실행하기

```
[root@Dci gcc-3.2.1]# make
```

2.6.6. make install실행하기

make install실행으로 우리가 설정한 설치 디렉토리(/usr/local/arm)에 설치하도록 한다. 이 명령 수행후 해당 디렉토리에 디렉토리 및 파일들이 설치되어 있는가 확인하자.

```
[root@Dci gcc-3.2.1]# make install
...
[root@Dci gcc-3.2.1]# cd /usr/local/arm/bin
[root@Dci bin]# ls -al
합계 18228
drwxr-xr-x  2 root    root      4096  7월 19 09:57 .
drwxr-xr-x 12 root    root      4096  7월 18 21:50 ..
-rwxr-xr-x  1 root    root    1207098  7월 18 21:06 arm-linux-addr2line
-rwxr-xr-x  2 root    root    1188036  7월 18 21:06 arm-linux-ar
-rwxr-xr-x  2 root    root    1827911  7월 18 21:06 arm-linux-as
-rwxr-xr-x  2 root    root    162265  7월 19 09:56 arm-linux-c++
-rwxr-xr-x  1 root    root    115920  7월 19 09:56 arm-linux-c++filt
-rwxr-xr-x  1 root    root    162292  7월 19 09:57 arm-linux-cpp
-rwxr-xr-x  2 root    root    162265  7월 19 09:56 arm-linux-g++
-rwxr-xr-x  1 root    root    157933  7월 19 09:57 arm-linux-gcc
-rwxr-xr-x  1 root    root     16396  7월 19 09:57 arm-linux-gccbug
-rwxr-xr-x  1 root    root     44588  7월 19 09:57 arm-linux-gcov
-rwxr-xr-x  2 root    root    1796559  7월 18 21:06 arm-linux-ld
-rwxr-xr-x  2 root    root    1232243  7월 18 21:06 arm-linux-nm
-rwxr-xr-x  1 root    root    1574180  7월 18 21:06 arm-linux-objcopy
-rwxr-xr-x  1 root    root    1659881  7월 18 21:06 arm-linux-objdump
-rwxr-xr-x  2 root    root    1188031  7월 18 21:06 arm-linux-ranlib
-rwxr-xr-x  1 root    root    284928  7월 18 21:06 arm-linux-readelf
-rwxr-xr-x  1 root    root    1112473  7월 18 21:06 arm-linux-size
-rwxr-xr-x  1 root    root    1096875  7월 18 21:06 arm-linux-strings
-rwxr-xr-x  2 root    root    1574179  7월 18 21:06 arm-linux-strip
-rwxr-xr-x  1 root    root      3371  7월 18 21:49 catchsegv
-rwxr-xr-x  1 root    root     74065  7월 18 21:49 gencat
```

```

-rwxr-xr-x 1 root root 42339 7월 18 21:49 getconf
-rwxr-xr-x 1 root root 63692 7월 18 21:49 getent
-rwxr-xr-x 1 root root 7453 7월 18 21:48 glibcbug
-rwxr-xr-x 1 root root 222683 7월 18 21:48 iconv
-rwxr-xr-x 1 root root 4613 7월 18 21:50 ldd
-rwxr-xr-x 1 root root 139819 7월 18 21:49 locale
-rwxr-xr-x 1 root root 968594 7월 18 21:49 localedef
-rwxr-xr-x 1 root root 6434 7월 18 21:49 mtrace
-rwxr-xr-x 1 root root 36648 7월 18 21:49 pcprofiledump
-rwxr-xr-x 1 root root 263849 7월 18 21:49 rpcgen
-rwxr-xr-x 1 root root 74608 7월 18 21:50 sprof
-rwxr-xr-x 1 root root 6923 7월 18 21:49 tzselect
-rwxr-xr-x 1 root root 5231 7월 18 21:49 xtrace

```

아래에 설치된 실행파일에 대한 간단한 설명을 참조하라.

실행파일 이름	설 명
arm-linux-gcc	GCC 컴파일러
arm-linux-c++	C++ 컴파일러
arm-linux-as	GNU 어셈블러
arm-linux-gasp	GNU 어셈블러 프리프로세서
arm-linux-ld	GNU 링커
arm-linux-objdump	오브젝트 정보 표시
arm-linux-strip	오브젝트 파일 심볼정보 제거
arm-linux-objcopy	오브젝트 파일 복사변환

2.7. 컴파일러 테스트하기

테스트 하기 위하여 우선 아래와 같은 파일을 하나 만들자.

```

[root@Dci src]# cat > test.c
#include <stdio.h>
int main()
{
    printf("Hello World!\n");
}
[Ctrl]-[D]키를 눌러 파일종료를 알린다.
[root@Dci src]# ls
test.c

```

이제 위에서 만든 파일을 크로스컴파일러로 컴파일 한다.

```

[root@Dci src]# arm-linux-gcc -o test test.c

```

위의 명령을 해서 에러가 없이 test 실행파일이 생성되었다면, 생성된 test파일의 정보를 아래와 같이 file 명령을 사용하여 종류를 확인하다.

```
[root@Dci src]# file test
test: ELF 32-bit LSB executable, ARM, version 1 (ARM), dynamically linked (uses shared libs),
not stripped
```

위에서 보면 test 파일은 ELF 32타입으로 실행파일이고, ARM용이라는 정보가 출력된다면 크로스 컴파일러가 제대로 설치된 것이다.

3. 개발보드를 위한 네트워크 설정

우선 타겟보드가 부팅이 완료되고, 네트워크를 위한 모든 준비가 되어 있다는 전제로 설명하도록 하겠다. 이 책을 읽는 개발자가 적어도 하나의 상용 개발보드를 구비하고 있다고 생각이 되고, 그렇다면 우선 개발보드를 가지고 이것 저것 해보는 것이 흥미를 유도하기에 좋을 것이기 때문이다.

3.1. tftp 사용하기

TFTP (Trivial File Transfer Protocol) 은 FTP와 같이 파일 전송을 하기는 하지만, UDP를 사용하고 간략하다는 장점이 있는 반면 보안에는 취약하다는 단점을 가진다. 하지만, 개발용으로써 이것이 FTP보다 용이하다는 점, 간단하다는 점등의 장점을 이유로 많이 사용된다. 특히 부트로더상에서 컴파일된 커널이미지를 네트워크로 부터 다운로드 받는 등의 일을 할 때 사용되어 진다. 물론 [JTAG]를 사용하여 직접 써 넣을 수도 있겠지만, [JTAG]의 속도가 무척 느리다는 점을 생각한다면 랜을 통하여 TFTP를 사용하는 다운로드 방법을 이용하게 된다. 이러한 작업을 하기위 하여서는 호스트컴퓨터에 TFTP를 사용하기 위한 설정이 되어 있어야 하며 여기서는 이 설정작업을 보인다. 설정에 들어가기전에 우선 호스트의 /usr/sbin/in.tftpd 데몬이 존재하는가 확인하자. 없다면 <http://www.rpmfind.net>에서 tftp-server RPM을 찾아서 설치하도록 하라.

3.1.1. 호스트 컴퓨터의 설정

아래에서는 슈퍼데몬 Xinetd의 관리에 들어가게 하기 위하여 TFTP 데몬을 활성화한다.

3.1.1.1. /etc/xinetd.d/tftp파일을 만들고 아래와 같이 만들고 xinetd데몬을 재실행한다.

```
[root@Developers xinetd.d]# pwd
/etc/xinetd.d
[root@Developers xinetd.d]# cat tftp
service tftp
{
```

```

socket_type      = dgram
protocol         = udp
wait             = yes
user             = root
server           = /usr/sbin/in.tftpd      #tftp 데몬파일
server_args      = -s /tftpd            #tftp를 하기위한 디렉토리
disable          = no                    #허용
per_source       = 11
cps              = 100 2
flags            = IPv4
}
[root@Developers xinetd.d]# /etc/rc.d/init.d/xinetd restart
Stopping xinetd:
Starting xinetd:
[ OK ]
[ OK ]

```

위에서 설정 내용중에서 server_args 부분에 tftp를 위해 사용될 디렉토리를 주어 준다. 여기서는 /tftpd 디렉토리를 만들고 이 디렉토리를 tftp를 위해 사용하고 있다.

3.1.1.2. 데몬이 실행중인지 확인

```

[root@DciDevelop xinetd.d]# netstat -a |grep tftp
udp        0      0 *:tftp          *.*

```

3.2. nfs 사용하기

NFS(Network File System)은 썬마이크로시스템스사가 개발한 것으로 원격지의 컴퓨터에 있는 파일을 자신의 컴퓨터에 있는것 처럼 사용할 수 있도록 한다.

임베디드 시스템에서 디바이스드라이버 또는 응용프로그램을 실행을 시켜보고자 할 때 하나 하나하면서 타겟보드에 FTP등을 이용하여 파일을 복사하고 사용하기는 힘들다. 더군다나 램디스크에 포함시켰다 지웠다 하는 작업은 더욱 힘들다.

이런 작업들의 편리성을 위하여 NFS(Network File System)을 사용하는데 이 NFS를 사용하게 되면 네트워크 상의 호스트컴퓨터의 하나의 디렉토리를 Export하여 타겟보드에서 이 디렉토리를 마운트하여 호스트컴퓨터의 파일 및 디렉토리에 대해 실행, 삭제, 수정을 할 수 있다.

이 NFS는 타겟보드에 대한 LINUX 포팅을 다 마치고, 로그인 할 수 있는 후의 작업이다. 여기서는 이 NFS를 사용하기 위하여 설정해야 하는 작업을 설명한다.

3.2.1. 호스트 컴퓨터의 설정

■ 공유할 디렉토리 설정하기

공유디렉토리를 만들고 공유할 디렉토리를 export하기 위하여 다음 /etc/exports 파일을 수정한다. 형식은 [공유할 디렉토리] [공유하여 사용할 머신의 IP : 여기서는 보드 IP](rw(읽기쓰기),권한) 형식으로 사용된다. 그 다음 /etc/init.d/nfs 실행파일을 재기동한다.

```
[root@Developers etc]# mkdir /nfs_share
[root@Developers etc]# pwd
/etc
[root@Developers etc]# cat exports
/root/nfs_share 192.168.0.99(rw,no_root_squash)
[root@Developers etc]# /etc/init.d/nfs stop
Shutting down NFS mountd: [ OK ]
Shutting down NFS daemon: [ OK ]
Shutting down NFS services: [ OK ]
Shutting down NFS quotas: [ OK ]
[root@Developers etc]# /etc/init.d/nfs start
Starting NFS services: [ OK ]
Starting NFS quotas: [ OK ]
Starting NFS mountd: [ OK ]
Starting NFS daemon: [ OK ]
[root@Developers etc]#
```

3.2.2 보드에서 호스트 디렉토리 마운트하기

make menuconfig 설정의 File System 설정중 Network File system (NFS) Support를 체크하여 컴파일 한다.

위에서 공유한 호스트의 디렉토리(/nfs_share)를 마운트하여 사용해 보자.

명령형식은 mount <호스트PC IP>:<공유 디렉토리> <마운트할 위치> 형식이다.

```
mount -t nfs 192.168.0.10:/nfs_share /mnt
```

관련정보

▣ 다음과 같은 에러가 난다면

```
[root@Linux /sbin]$mount 218.53.171.50:/home/sohnet /usr/mnt
```

```
nfs warning: mount version older than kernel
```

```
lockd_up: no pid, 2 users??
```

```
RPC: sendmsg returned error 101
```

☞ mount -t nfs -o nolock xxx.xxx.xxx.xxx:/nfsd /mnt/nfsd 와 같은 옵션으로 실행하여 보자.

▣ nfs가 동작을 안하는 것 같은 경우

☞ 램디스크에 /sbin/portmap실행파일이 없어서 일수 도 있다. 이 문제는 다른 램디스크를 구해서 복사하고 실행하라.

☞ 커널컴파일시에 file system-> nfs support를 설정하였는가?

nfsv3 client support를 설정하였는가

☞ BusyBox로 램디스크에 들어갈 파일을 만들때 Config.h파일에서 Mount/UnMount명령뿐 아니라 NFSMount옵션을 선택하였는가? (램디스크 만들기 참조)

3

JTAG에 대하여

JTAG(Joint Test Access Group)는 무엇이고 언제 사용되는가? 왜 알아야 하는가?

JTAG는 " 보드의 하드웨어적 디버깅 하기위한 막강한 툴이다" 라고 말할 수 있다. 물론 개발보드업체에서 제공하는 JTAG툴을 사용하거나, 인텔사에서 공개하고 있는 툴을 사용하거나, <http://sourceforge.net> 사이트에서 프로젝트로 진행중인 Jtag를 사용할 수 있지만, 소스를 자신의 시스템에 맞게 수정한다든지 해야 하는 경우가 생기므로 JTAG의 동작원리를 아는 것은 중요할 수 있다.

JTAG의 발생배경, 강력함을 알기 위하여 JTAG가 탄생하기전의 보드의 하드웨어적인 디버깅을 생각해보자. 경제적으로 부유한 회사에서는 지그라는 장비(해당 하드웨어의 디버깅 포인트마다 못을 박아 놓은 형태의 것으로 하드웨어 디버깅용으로 사용하며, 못을 받아 놓은 모양이라서 탐침(Bed of Nails)이라고도 한다.)를 만들어 하드웨어 디버깅을 하겠지만, 그렇지 못한 회사에서는 오실로스코프가 다 경우가 허다하다. 실제 필자도 그런 회사에 있다. 하드웨어 설계를 완벽히 하고, PCB 아트워킹을 완벽히 하면 되겠구나 생각들 하겠지만, 하드웨어 엔지니어들에게 이런 이야기를 한다면 가당치 않은 이야기라며 혀를 내두를 것이다. 남뎀 부터, 부품소자의 특성등에 의한 사소한 문제가 시스템 전체의 동작에 영향을 끼치게 되면 무척 당황되고, 이를 찾기 위해 몇 일 밤낮을 지새우는 경우가 실제로 빈번하다. 하지만, 막상 원인을 찾았을 때 큰 문제가 아닌 사소한 문제인 경우가 많다. 이러한 어려움뿐만 아니라 IC등의 핀수가 점차 많아 지고, 타입이 DIP에서부터 PGA, SOIC, BGA등의 다양한 모습으로 바뀌면서 핀사이의 간격이 좁아지는 것뿐만 아니라, PCB의 경우에서도 단층, 2층, 4층, 8층등의 발전으로 오실로스코프만으로 또는 지그장비로의 디버깅 작업은 힘이 들며, 분명 한계가 들어난다.

이런 어려운 하드웨어 디버깅을 위하여 "원하는 핀에 원하는 시그널을 낼 수 있으면, 하드웨어 디버깅이 얼마나 편해지겠는가?" 생각해 보라. 어떤 IC가 잘못되었는지 배선이 어느곳이 잘못되었는지 등을 쉽게 해결할 수 있을 것이다.

이런 초기 아이디어로 JTAG는 1980년대 중반에 만들어 졌으며 1990년에는 IEEE의 표준이 되었다. 이 JTAG의 정확한 명칭은 IEEE 1149.1또는 Boundary Scan

Circuit이다. 하드웨어 디버깅에서 JTAG를 사용하면, 배선이 잘못되었는지, 신호가 제대로 나오고 동작을 하는지를 알기 위하여 특정 핀에 특정 출력을 내 줄수 있으므로 회로의 동작상태를 쉽게 확인가능하다.

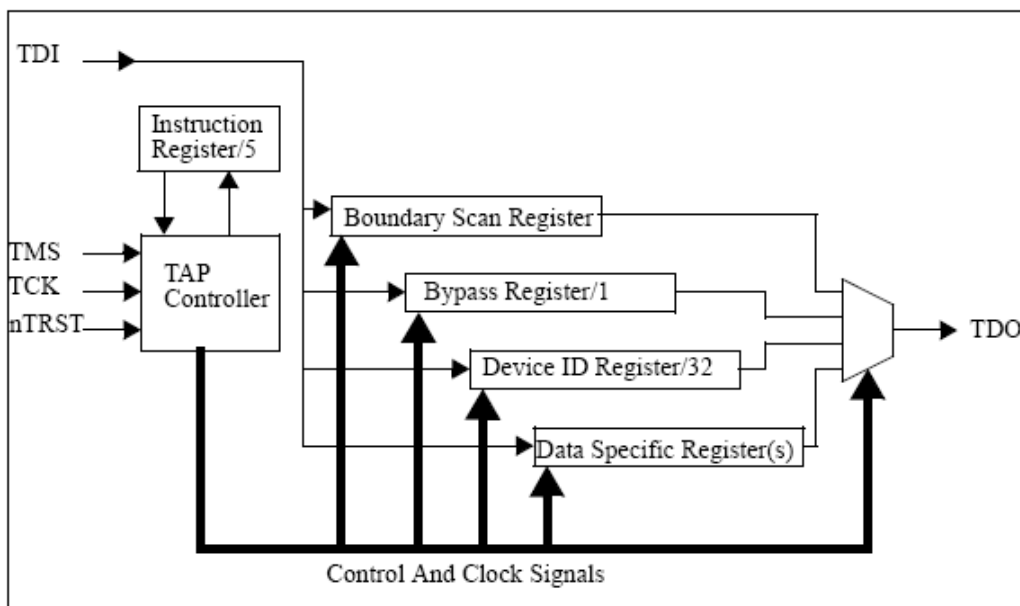
프로그램 코드가 들어갈 메모리의 경우를 보면 플래시 롬을 많이 사용되는데, 보통 SMD 타입의 플래쉬 롬을 사용하여 PCB에 소켓없이 장착하게 되므로 롬라이터를 이용하여 굽기란 힘든 일이다. 이러한 작업을 JTAG를 사용하여 Address버스와 Data버스, R/W/CS 시그널핀들에 적당한 시그널을 줌으로써 보드에 붙어 있는 상태로 플래시에 부트스트랩등의 프로그램을 써 넣을 수 있다. 실제로 이 작업에 많이 사용하게 될 것이다.

그럼 이제 JTAG의 구조에 대하여 알아보도록 하자. 이 JTAG(Joint Test Access Group)란 말이 나오면 Boundary-Scan 이라는 말이 꼭 나온다.

우선 이 Boundary-Scan의 구조와 원리를 알아보고 TAP Controller에 대하여 알아볼 것이며, 실제 프로그램 순서를 보면서 원리를 이해해 나가도록 한다.

1. JTAG의 구조

아래 그림에서 JTAG의 전체적인 블록도를 보인다.



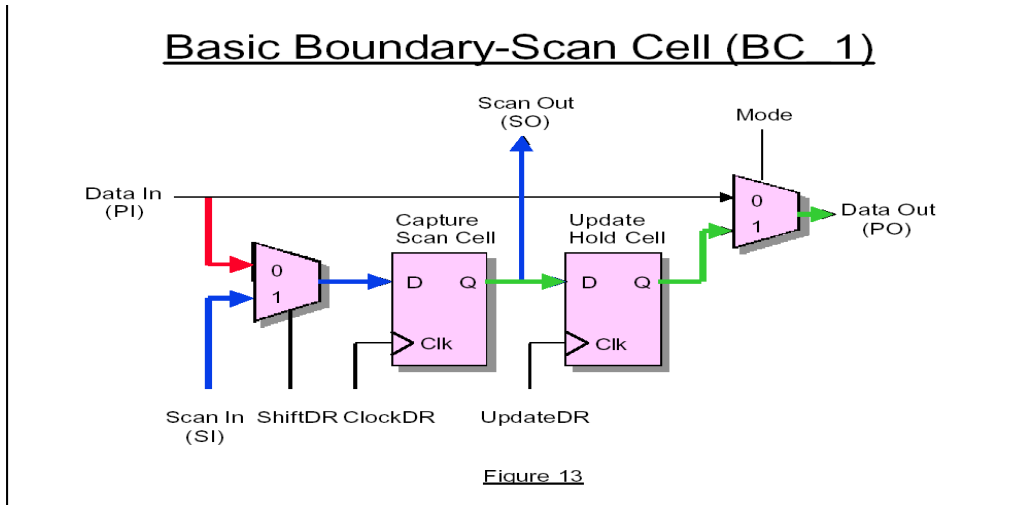
1.1. BSC(Boundary-Scan Cell)의 구조

위의 그림을 보면 Boundary-Scan은 Boundary-Scan Cell이라는 구성요소로 구성됨을 볼 수 있다.

이 BSC는 입/출력 핀들에 맵핑되어 존재하며, 전원핀 및 기타 특정핀들은 제외된

다. BSC의 수는 핀의 종류에 따라 입력, 출력, 입출력, 3-State이 존재하기 때문에 핀 수와 정확히 일치하는 숫자를 가지고 있지는 않다. 입/출력 핀을 보자면, 우선 입력 셀, 출력셀, 그리고 입/출력 방향을 정하는 셀로 3개가 존재하게 된다.

이 구성요소인 BSC의 구조는 아래와 같다.



우선 CELL의 외부에는 크게 PI(Parallel Input), PO(Parallel Output), SI(Serial Input), SO(Serial Output)를 가지고 있다. 아래 표에서 이를 설명한다.

이름	동작
PI	외부 입력핀 또는 내부코어의 출력에서 받아들이는 시그널
PO	셀의 값을 외부 출력핀 또는 내부코어의 입력으로 출력
SI	직렬 입력포트(TDI핀) 또는 이전 셀의 신호로 연결
SO	직렬 출력포트(TDO핀) 또는 다음 셀의 SI와 연결

이 셀의 내부 기능모드에는 Normal Mode, Update Mode, Capture Mode, Shift Mode의 4가지 모드가 있으며 각각의 동작은 아래 표와 같다.

모드	동작
Normal Mode	PI(Data_In)가 PO(Data_Out)으로 곧바로 나간다.
Update Mode	Update Hold Cell의 내용이 PO로 나간다.
Capture Mode	PI(Data_In)의 시그널이 Capture Scan Cell의 입력이 되어 ClockDR에 의해서 캡처되어 진다. 이 ClockDR은 TCK핀에서 나온다.
Shift Mode	하나의 Capture Scan Cell의 SO(Scan_Out)는 다음의 Capture Scan Cell의 SI(Scan_In)로 연결된다

위의 동작들은 JTAG를 다루면서 이해하게 될 것이므로 지금 당장 고심할 필요는 없다. 사실 이런 셀 내부의 동작은 우리가 직접 관여하지는 할 필요는 없는 것이며, 아래 설명하게 될 JTAG 외부핀과 TAPC 동작을 이해하고 사용하면 된다.

1.2. JTAG 외부핀

위에서는 JTAG의 전체적인 모양과 그 구성요소인 셀에 대하여 간략하게 살펴 보았다. 이제 JTAG를 위한 외부 핀에 대하여 알아 볼 차례이고, 우리가 실제로 직접 대면하게 되는 것은 내부 셀이 아니라 이 핀들이 될 것이다.

그림을 보면 외부적으로는 JTAG를 지원하는 칩의 Test Data In(TDI), Test Mode Select(TMS), Test Clock(TCK), Test Data Out(TDO)이 존재하는데 이들 핀들은 JTAG를 지원하는 IC에서는 IEEE 표준에서 꼭 있어야 하는 핀으로 규정하고 있고, 부가적으로 달릴수 있는 핀으로는 Test Reset(TRST)이 있다. 이들 핀들을 Test Access Port(TAP)라고 부른다. JTAG지원 IC들은 내부적으로는 Instruction Register, 1-bit Bypass Register를 가지고 있어야 하며, 옵션으로 32-bit Identification Register, TAP Controller를 가지는 데 특히 TAP controller를 주의 깊게 보기 바란다. 우리는 위에서 본 외부 핀들(TAP)을 통해 데이터를 주어 이 컨트롤러의 상태를 제어함으로써 JTAG에 대한 모든 제어를 할 것이다.

아래 표에서 각 핀들이 하는 역할을 보인다.

핀 이름	방 향	동 작
TDI	입 력	직렬 입력포트로 셀의 SI로 연결된다. TCK신호의 Rising Edge에서 전달된다.
TDO	출 력	직렬 출력포트로 셀의 SO로 연결된다. TCK신호의 Falling Edge에서 전달된다.
TCK	입 력	시그널들의 동기화를 위한 클럭신호이다.
TMS	입 력	TAP 컨트롤러에서의 모드를 선택하기 위한 제어시그널이다. TCK의 Rising Edge에서 이 핀의 상태에 따라 다음 상태로의 전이가 일어난다.
TRST	입 력	TAP 컨트롤러의 리셋신호이다.

전체적인 동작에 대해서 우선 간략히 보자면 명령을 주기 위하여 TMS 시그널을 통하여 TAP 컨트롤러의 상태를 Instruction Register(IR)를 선택하는 모드로 전이하며, Instruction Register(IR)로 TDI핀 시그널을 통하여 명령코드 값을 TAP Controller에 전달한다. 그 이후 해당 Data Register(DR) 레지스터를 선택하고, TDI를 통하여 이 Data Register(DR)에 값을 써 넣어 셀들의 상태를 변경시키거나 TDO를 통하여 데이

터를 읽어 옴으로써 셀들의 상태를 알아오는 작업을 통하여 우리가 원하는 일을 수행할 수 있는 것이다.

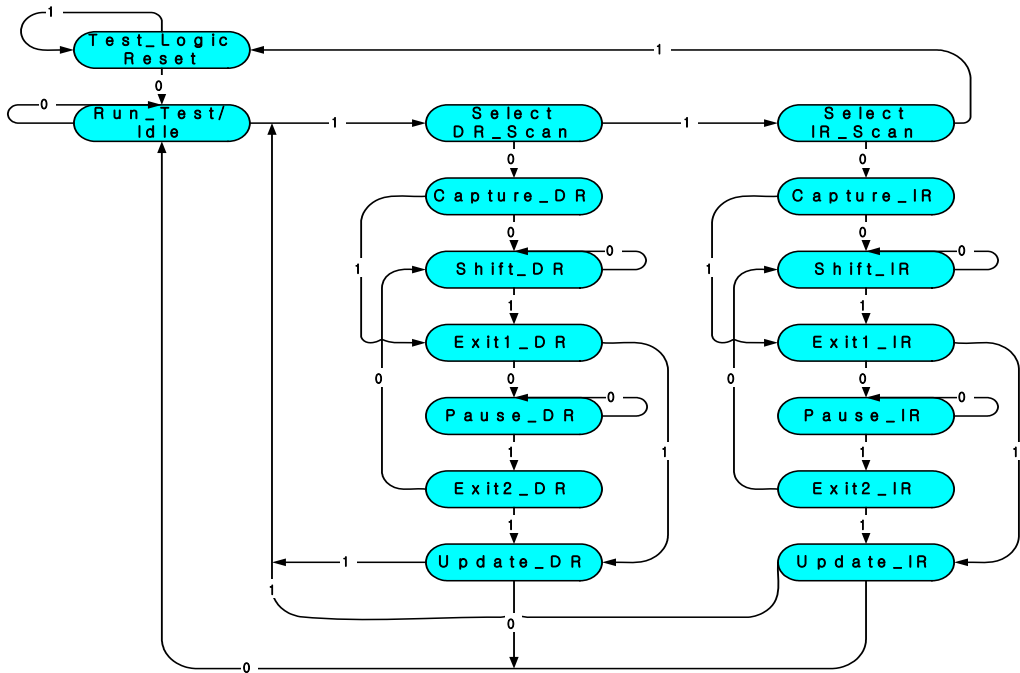
JTAG의 TAP 컨트롤러(TAPC)를 제어하기 위하여 Instruction Register(IR)의 명령 코드들을 알아야 한다. Instruction Register에 써넣을 수 있는 몇가지 정의된 명령값을 가지고 있으며, 이는 4비트가 될수도 5비트가 될 수도 있다. PXA-255는 5비트의 JTAG 명령으로 구성되어 있다.

아래 보이는 명령들은 우리가 자주 쓰게 될 PXA-255의 JTAG 명령들을 나열한 것이며, 이 명령비트들은 아키텍처마다 다를 수 있다.

명 령	명령비트	설 명
EXTEST	00000	이 명령은 핀의 상태를 읽어오거나, 핀의 상태를 제어하기 위한 명령으로 우리는 보드 테스트, JTAG ID 읽기, 플래쉬에 쓰기등에 이 명령을 사용한다.
BYPASS	11111	ByPass레지스터는 TDI의 신호가 TDO 바로 나간다. 이 모드는 여러개의 JTAG 칩들을 시험할 때 시험하지 않는 칩에 대하여 건너뛰기 역할을 한다고 보면 되겠다.
IDCODE	00111	ID CODE를 갖는 ID 레지스터에서 디바이스의 ID를 얻는데 사용된다.
HIGHZ	01000	모든 핀들을 하이 임피던스 상태로 만든다. 이렇게 함으로써 시험하는 칩의 외부에 연결된 다른 칩들의 동작을 시험하고 할 때 이 칩이 외부시그널의 영향을 받지 않고 또한 외부에 영향을 주지 않도록 하기 위하여 사용된다.

2. TAPC(TAP Controller)에 대해

위에서 JTAG를 사용하기 위한 전반적인 작업이 TAPC의 상태전이를 통하여 이루어진다고 했다. 아래 전체적인 상태전이 도를 보인다.



모 드	설 명
Test_Logic Reset	TAPC 및 JTAG의 모든 내용을 초기화한다.
Run-Test/Idle	JTAG를 동작의 초기점이다.
Select DR-Scan	SHIFT_DR로 가기위한 상태전이 중간단계로 하는 일은 없다.
Capture_DR	PI 값을 SHIFT하기 위한 준비단계로 하는 일은 없다.
Shift_DR	PI 값을 IR 레지스터에 의해 선택된 내부 레지스터에 써 넣는다. (TDI의 값을 TCK시그널에 맞춰 써넣는다.)
Exit1_DR	상태전이 중간단계로 하는 일은 없다.
Pause_DR	상태변이 중간단계로 하는 일은 없다.
Exit2_DR	상태변이 중간단계로 하는 일은 없다.
Update_DR	이 단계에서 위에서 PI 값을 IR 레지스터에 의해 선택된 내부 레지스터에 써 넣은 값을 PO에 적용시킨다.
Select IR-Scan	SHIFT_IR로 가기위한 상태전이 중간단계로 하는 일은 없다.
Capture_IR	PI 값을 SHIFT하기 위한 준비단계로 하는 일은 없다.

Shift_IR	PI 값을 IR 레지스터에 써넣는다. (위에서 본 작업하고자 하는 명령 코드를 TDI의 값을 TCK신호에 맞춰 써 넣는다.)
Exit1_IR	상태전이 중간단계로 하는 일은 없다.
Pause_IR	상태변이 중간단계로 하는 일은 없다.
Exit2_IR	상태변이 중간단계로 하는 일은 없다.
Update_IR	이 단계에서 위에서 써넣은 명령코드를 IR 레지스터에 적용시킨다.

위의 그림처럼 TAP controller는 16개의 상태테이블을 가진다. 상태전이에 나타난 1, 0들의 값이 나타내는 것은 TMS의 값을 의미하며, TCK신호가 Low->High로 되면서 적용이 된다.

이 후 과정을 따라가기에 앞서 우리가 프로그래밍을 할 때 고려해야할 몇가지 규칙을 정리해 보자.

순서	내 용
1	TAP 컨트롤러가 초기화되면 Test_Ligic_Reset상태로 들어온다. 이때 TMS가 1로 되어 있는 동안 상태가 변하지 않는다.
2	TAP 컨트롤러의 어떤 상태에 있더라도 TMS가 1인 상태로 6클럭이상만 주면 Test_Logic Reset 상태로 돌아온다
3	Shift_IR상태에서 TMS=0인 상태를 유지하면서 Instruction Register에 TDI를 통하여 값을 써 넣어주며, Shift_DR 상태에서 TMS=0인 상태를 유지하면서 TDI를 통해 값을 넣거나 TDO를 통해 값을 읽어온다. 이때 TDI를 통해 값을 넣을때는 TCK가 Low->High로 될 때 넣어지고, TDO를 통해 읽어 올때는 High->Low로 될 때 데이터가 나온다. 다음 상태로 넘어가기 위해서는 TMS=1인 상태에서 Low->High로 될 때 넘어가므로, Shift_IR상태나 Shift_DR상태에서 TDI를 통해 값을 넣을 때에 총비트-1의 비트를 써 넣어 주고, 다음 상태로 전이하면서 마지막 한비트를 써 넣어주어야 한다
4	명령코드를 써 넣을 때는 하위비트부터 써 넣는다. 즉 ID코드 명령인 00111은 출력을 할때 11100의 순서로 넣어주어야 하므로 1->1->1->0을 Shift_IR상태에서 넣어준 다음 다음 상태로 넘어가면서 마지막 비트인 0을 써주어야 한다
5	Test_Logic Reset 상태로 가면 Shift_IR에서 이전에 넣어준 명령코드도 또한 리셋되므로 명령코드가 유효하기를 바란다면 Run_Test/Idle상태로 가야 한다. 여섯째, 만약 EXTEST 명령등으로 출력과 관련된 명령은 Update_DR상태로 되면서, 셀들의 값이 핀으로 출력되어 진다. 읽을 때도 역시 Capture_DR상태로 되면서, 핀들의 값을 각 셀들로 읽어진다.

3. ID읽기와 외부핀 제어하기

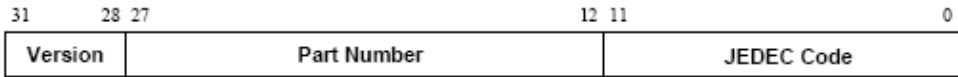
위에서 우리는 JTAG에 대한 개략적인 정의와 TAP 컨트롤러의 상태도를 보았다. 이 장에서는 아래 단계적인 작업과정을 통하여 이들에 대한 이해를 돕도록 한다.

3.1. ID Code 읽어오기

PXA-255 JTAG에는 PXA-255임을 나타내는 ID코드를 가지고 있다. 이 장에서는 이 ID코드를 읽어오기 위한 방법을 살펴본다.

3.1.1. PXA-255 ID 코드

Xscale ID Register의 구조는 다음과 같다.

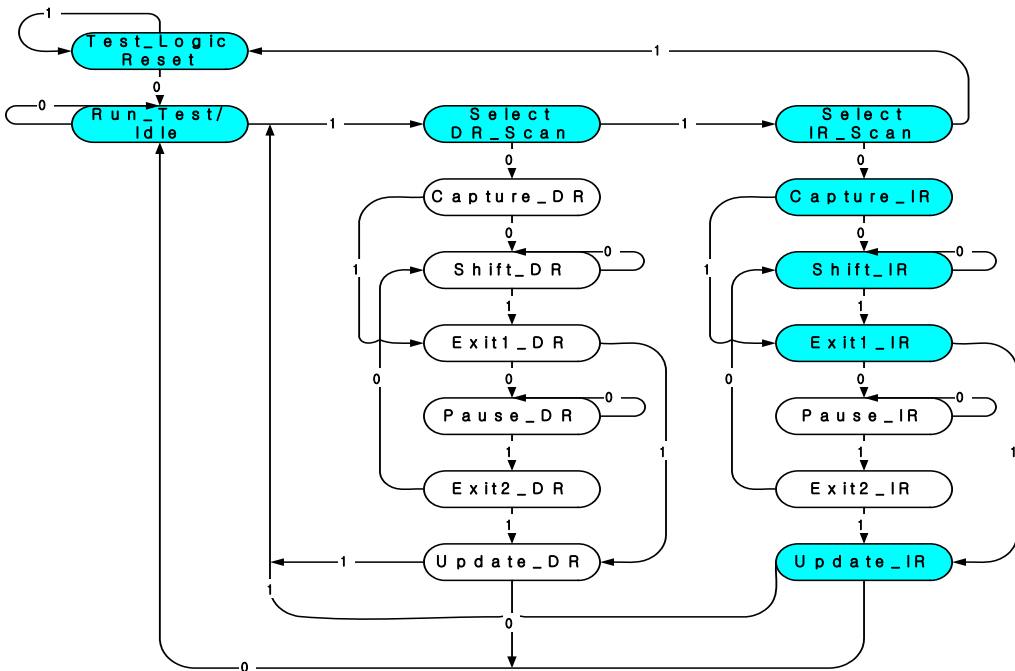


PXA255의 JTAG 값은 Stepping Revision A0인 경우 0x6926_4013 값을 갖고 있으며, 우리는 JTAG ID를 읽어 이 값과 비교하여 PXA255인가를 판별하면 되겠다.

3.1.2. ID Code 읽기 수행과정

ID를 읽어 오려면, Shift_IR에 ID_CODE Instruction(00111)을 써 준 다음, Shift_DR 상태에서 데이터를 읽어 와야 한다. 초기상태는 Test_Logic Reset 상태이며, 코드를 읽어 오기 위한 작업을 위하여 아래의 TAP controller 상태도에서 색깔이 있는 곳을 따라가 보자. 숫자로 값이 표기된 것은 TMI의 값이며, 이 TMI는 TCK의 Rising Edge 즉, Low -> High로 될 때 상태가 변한다.

3.1.2.1. Shift IR에 명령코드 쓰기



1. Shift_IR 상태로 간다. (처음 상태는 Test_Logic Reset상태로 생각한다.)

Run_Test/Idle 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1
▼	
Select DR_Scan 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
▼	
Select IR_Scan 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
▼	
Capture_IR 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1
▼	
Shift_IR 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1



2. ID_CODE Instruction을 넣는다. (TMS=0)

여기서 주의해야 할 점은 LSB부터 넣는다는 것이다. (00110) 또한, 다음 상태로 전이하는데 TCK가 0→1로 신호가 들어가게 되므로 마지막 비트는 다음 상태로 전이하면서 인가해야 한다. 즉, 여기서는 4비트를 그리고, 전이를 하는 신호와 함께 1비트를 주어야 한다. 이렇게 하지 않고 여기서 5비트를 다준다면, 다음 상태로 전이를 하면서 엉뚱한 값이 쉬프트되어 들어가게 되므로 주의하자.

"0"	TCK=0 ▷ TDI=0 ▷ TCK=1
▼	
"1"	TCK=0 ▷ TDI=1 ▷ TCK=1
▼	
"1"	TCK=0 ▷ TDI=1 ▷ TCK=1
▼	
"0"	TCK=0 ▷ TDI=0 ▷ TCK=1

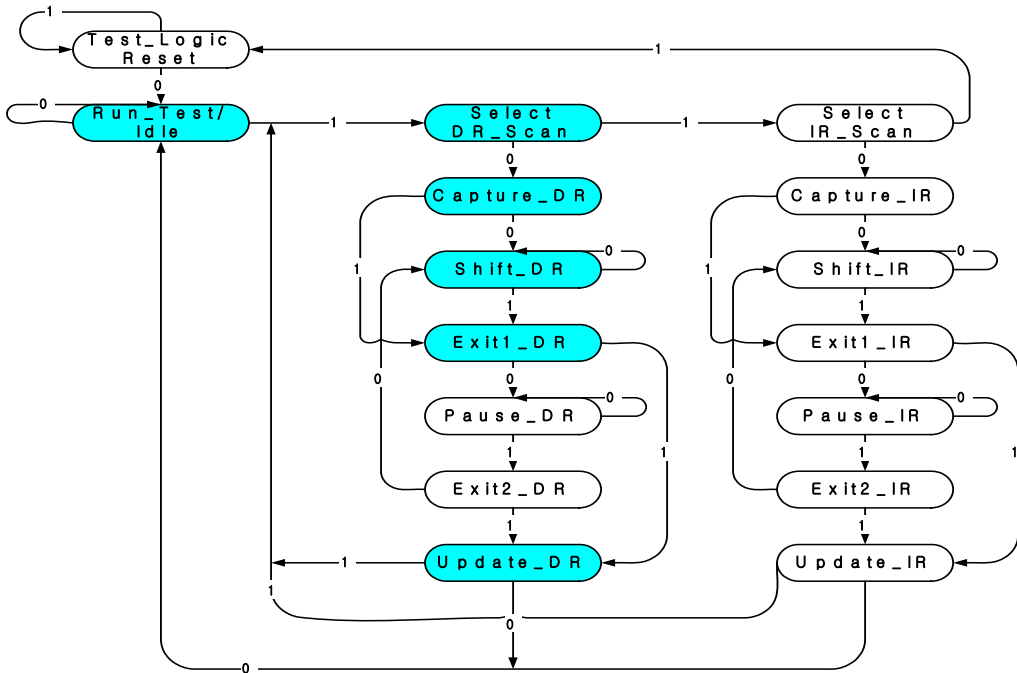


3. ID_CODE의 마지막 비트를 주고, Run_Test/Idle상태로 전이한다.

ID_CODE를 마지막 비트 "0"을 쓰면서 Exit_IR상태로 전이	TCK=0 ▷ TDI=0 ▷ TMS=1 ▷ TCK=1
▼	
Update_IR 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
▼	
Run_Test/Idle 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1

3.1.2.2. Shift_DR 에서 ID Code 읽기

위에서 우리는 Shift_IR상태에서 00111을 Instruction Register에 넣어줌으로써 ID Register를 선택하였으므로 여기서는 Shift_DR상태에서 값을 읽어 오면 된다. 위에서 Run_Test/Idle상태로 마쳤다. 만약 Test_Logic Reset 상태로 간다면 위에서 넣어준 ID_CODE명령도 리셋되므로 주의하자.



1. 위에서 마친 Run_Idle 상태에서 TAPC상태를 Shift-DR상태로 한다.

Select DR_Scan 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
-----------------------	-----------------------



Capture DR 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1
-------------------	-----------------------



Shift DR 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1
-----------------	-----------------------



2. ID CODE 패턴을 읽어 온다. (TMS=0)

여기서 부터 TCK가 High->Low로 되면서 TDO로 한비트씩 값이 나온다.위에서 TCK가 1로 끝나 있음을 주목하다. TMS와 TDI은 0을 유지하면서 32비트를 읽어 오면 된다.

```

for (i=31 ; i>=0; i--) {
    TCK=1;
    ID_CODE[i]= TDO;
    TCK=0;
}
  
```



3. TAPC상태를 Run-Test/Idle상태로 만든다.

Exit1-DR 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
-----------------	-----------------------



Update-DR 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
------------------	-----------------------



Run-Test/Idle 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1
----------------------	-----------------------

3.2. 외부핀 제어하기

3.2.1. BSDL에 대하여

외부핀을 제어하기 전에 PXA255의 셀과 해당 핀들간의 관계를 알아야 한다. 이런 관계를 표현하기 위하여 JTAG를 지원하는 디바이스들의 셀을 서술하기 위하여 각 디바이스업체들은 BSDL(Boundary Scan Description Language) 언어를 사용한 BSDL 파일을 제공한다. 아래 부록CD에 있는 PXA255 BSDL 파일중 일부분을 발췌하였다.

```

■ 디바이스 이름
entity pxa255_jtag_A0 is
generic(PHYSICAL_PIN_MAP : string := "MBGA_256");
■ 각 포트들에 대한 이름과 속성에 대한 정보
port(
    boot_sel      : in      bit_vector(2 downto 0);
    gpio          : inout   bit_vector(89 downto 0);
    dqm_0         : out     bit;
....
);
■ 패키지정보 및 이름과 핀번호간의 맵핑정보
gpio0라는 이름은 은 Pxa255에서 L10 Ball(Pin) 임을 나타낸다. 다음 gpio1번은 L12 Ball
이다.
use STD_1149_1_1994.all; -- Get IEEE 1149.1-1994 attributes and definitions
attribute COMPONENT_CONFORMANCE of pxa255_jtag_A0:entity is "STD_1149_1_1993";
attribute PIN_MAP of pxa255_jtag_A0 : entity is PHYSICAL_PIN_MAP;
constant MBGA_256 : PIN_MAP_STRING :=
    "gpio      : (D10,D3,F2,G3,F15,D16,E15,E16,F16,R13," &
        "T9,P9,A8,B8,D8,E8,B3,C3,A2,A3," &
        .....
        "F12,F14,G15,H14,J11,J12,K14,L13,L12,L10)," &
    .....
■ JTAG에 사용되는 핀들(CLK, TDI, TDO, TMS등), 명령코드길이, 명령코드종류에 대한
설명
attribute TAP_SCAN_IN of tdi : signal is true;
attribute TAP_SCAN_MODE of tms : signal is true;
attribute TAP_SCAN_OUT of tdo : signal is true;
attribute TAP_SCAN_CLOCK of tck : signal is (40.0e6, BOTH);
attribute TAP_SCAN_RESET of ntrst : signal is true;

attribute INSTRUCTION_LENGTH of pxa255_jtag_A0 : entity is 5;
attribute INSTRUCTION_OPCODE of pxa255_jtag_A0 : entity is
    "extest (00000)," &
    "bypass (11111)," &
    "sample (00001)," &
    "idcode (11110)," &
    "clamp (00100)," &
    "highz (01000)";
■ JTAG ID 설명.
attribute INSTRUCTION_CAPTURE of pxa255_jtag_A0 : entity is "00001";
attribute IDCODE_REGISTER of pxa255_jtag_A0 : entity is
    "0110" &                -- Version Number
    "1001001001100100" &    -- Part Number

```

```
"00000001001" &          -- Manufacturer ID
"1";                        -- Required by IEEE Std. 1149.1-1990
```

■ 접근할 내부 레지스터를 선택하기 위한 명령코드들

attribute REGISTER_ACCESS of pxa255_jtag_A0 : entity is

```
"BOUNDARY (extest, sample), " &
```

```
"DEVICE_ID (idcode), " &
```

```
"BYPASS (bypass, clamp, highz);
```

■ CELL의 정보

CELL의 총갯수가 410개 임을 나타내고 그 각 셀에 대한 아래 설명들을 한다.

아래는 GPIO0, GPIO1에 대한 것만 발췌하였다. GPIO포트는 입력과 출력 모두 가능하므로 입/출력 결정을 해주어야 한다. gpio(0)을 보면 ccell(control cell:입/출력제어)에 0번, disval(disable value:디스에이블(입력전환)되기 위한 값)에 0을, rslt(디스에이블시 셀상태)에 Z이 주어졌다. 이는 0번 Cell을 통하여 제어되고, "0"번 Cell에 1을 써 넣으면 인에이블 즉 출력으로 지정됨을 의미한다. 또한 "0"번 Cell에 0을 써 넣어 입력으로 전환시 이 Cell은 Z(하이 임피던스)상태가 됨을 의미한다.

즉, 우리가 gpio(0)번을 출력으로 High를 쓰고 싶다면, 우선 cell(0)에 1의 값을 주어 cell(95)를 출력상태로 만들고, cell(95)에 High(1)값을 쓰면 된다. 시그널을 읽을 때는 cell(0)을 0을 주어 cell(95)(gpio(0))의 출력상태를 디스에이블 시키고, cell(273)의 값을 읽으면 된다.

attribute BOUNDARY_LENGTH of pxa255_jtag_A0 : entity is 410;

attribute BOUNDARY_REGISTER of pxa255_jtag_A0 : entity is

```
-- num      cell      port  function safe [ccell disval rslt]
"0    ( bc_1,      *,    control,  0)," &
"1    ( bc_1,      *,    control,  0)," &
"95   ( bc_1,      gpio(0), output3,  X,    0,    0,  Z)," &
"96   ( bc_1,      gpio(1), output3,  X,    1,    0,  Z)," &
"273  ( bc_1,      gpio(0), input,    X)," &
"274  ( bc_1,      gpio(1), input,    X)," &
```

위의 BSDL 파일을 보고 셀의 종류와 핀의 상태를 알았다면 이제 EXTEST 명령 코드를 통하여 직접 핀들을 제어하고, 핀들의 상태를 읽어볼 수 있다.

3.2.2. 외부핀 제어 수행과정

외부핀 제어를 위하여 EXTEST명령을 사용한다. 우선 Shift_IR상태에서 EXTEST (00000) 명령값을 준 다음 Shift_DR상태에서 핀들의 각 상태를 TCK를 LOW->HIGH로 되면서 TDO로 값을 밀어넣고, TCK가 HIGH에서 LOW로 되면서 TDI로 부터 값을 읽어들이면 된다. 여기서는 Shift_IR상태, 로 가는법, Run_Idle상태로 가는 방법들은 위에서 보았으므로 설명하지 않을 것이다.

3.2.2.1. EXTEST 명령값 주기

1. Test_Logic_Reset으로 초기화 시킨다. (TMS=0)

상태도를 보면 TMS를 1상태로 6클럭이상을 주면 Test_Logic_Reset 상태로 초기화 된다.

```
for(i=0 ; i<6 ; i++) {
    TCK=0
    TMS=1
    TCK=1
}
```



2. TAPC상태를 Shift-IR상태로 한다.

Run_Test/Idle 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1
▼	
Select DR_Scan 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
▼	
Select IR_Scan 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
▼	
Capture_IR 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1
▼	
Shift_IR 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1



3. EXTEST 명령 패턴(00000)을 LSB부터 4비트만 넣는다. (TMS=0)

"0"	TCK=0 ▷ TDI=0 ▷ TCK=1
▼	
"0"	TCK=0 ▷ TDI=0 ▷ TCK=1
▼	
"0"	TCK=0 ▷ TDI=0 ▷ TCK=1
▼	
"0"	TCK=0 ▷ TDI=0 ▷ TCK=1



4. 마지막 EXTEST 비트를 쓰고 TAPC상태를 Run-Test/Idle상태로 만든다.

마지막 EXTEST명령 비트를 쓰고 Exit_IR 상태로 전이한다.	TCK=0 ▷ TDI=0 ▷ TMS=1 ▷ TCK=1
▼	
Update-IR 상태로 전이한다.	TCK=0 ▷ TMS=1 ▷ TCK=1
▼	
Run_Test/Idle 상태로 전이한다.	TCK=0 ▷ TMS=0 ▷ TCK=1

3.2.2.2. 핀제어/감시하기 (Shift DR상태에서 핀상태를 밀어 넣어 주면서 나온 값을 읽는다)

여기서 핀들에 대한 각 셀의 순서와 갯수에 대해서는 PXA255의 BSDL문서에 나오므로 참조하도록 하자. PXA255의 JTAG의 총 셀의 수는 410개로 되어 있으므로, 총 410개의 비트 패턴을 만들어서 SHIFT_IR에 순서대로 넣어 주어야 한다. 여기서 주의할 점은 밀려 나온 데이터는 지금 만들어 넣는 핀셀 배열에 따른 응답값이 아니라 이전의 핀 상태를 나타내는 값이라는 점이다. 여기서 밀어 넣어준 셀 값들이 핀에 적용되는 때는 Update_DR상태에서 이다.

1. 위에서 마친 Run_Idle 상태에서 TAPC상태를 Shift-DR상태로 한다.

Select DR_Scan 상태로 전이	TCK=0 ▷ TMS=1 ▷ TCK=1
▼	
Capture DR 상태로 전이	TCK=0 ▷ TMS=0 ▷ TCK=1

Shift DR 상태로 전이

TCK=0 ▷ TMS=0 ▷ TCK=1

2. 각 핀의 상태를 읽고 쓴다.

TMS의 값은 0으로 유지하면서 만든 비트 패턴 TCK가 High->Low 상태로 될때 TDO에서 값을 읽어오고 TCK가 Low->High로 되면서 TDI의 값을 읽어 각핀의 상태를 읽어 온다. 이것은 410번 반복한다. ID CODE읽어 오기에서도 말했지만, TDI로 값을 쓰는 것은 Low->High로 가면서 이루어 지고 다음 상태로 나가기 위하여서도 Low->High로 되므로 마지막 비트는 다음으로 넘어가면서 TDI에 값을 넣도록 한다.

이 부분에 대해서는 KelbJTag의 소스를 참조하기 바란다.

3. TAPC상태를 Run-Test/Idle상태로 만든다

Exit1-DR 상태로 전이

TCK=0 ▷ TMS=1 ▷ TCK=1

Update-DR 상태로 전이

TCK=0 ▷ TMS=1 ▷ TCK=1

Run-Test/Idle 상태로 전이

TCK=0 ▷ TMS=0 ▷ TCK=1

여기서 알아두어야 할것은 Update-DR상태에서 우리가 넣어준 비트패턴에 따라 핀들의 출력상태가 변하게 된다는 것이다

4. JTAG로 플래쉬메모리 접근하기

우리가 JTAG를 사용할 때는 하드웨어 테스트를 위한 것도 있지만, 플래쉬메모리에 데이터(부트로더등)을 쓰기 위하여 많이 사용된다. 여기서는 이런 작업을 위하여 플래쉬 메모리에 대한 전반지식과 어떻게 접근하는지에 대해 보도록 하겠다.

4.1. 플래쉬메모리에 대하여

EPROM은 비휘발성이고 빠른 읽기 속도와 집적도가 높지만 재 프로그램하기가 힘들다는 단점이 있고, EEPROM은 비휘발성이고 빠른 읽기 속도를 가지며 전기적으로 빠르게 삭제와 재프로그래밍이 가능하다는 장점이 있지만, 집적도가 낮고 제조 비용이 높다는 단점이 있다. 이에 비해 플래쉬 메모리는 불휘발성으로 EEPROM보다 집적도가 높고 집적도가 높아 상대적으로 작은 크기에 대용량이 가능하며 고속으로 데이터를 읽어 내기가 가능하고 전기적으로 삭제가 가능하다는 장점이 있어 임베디드 시스템에서 많이 쓰이고 있는 메모리이다.

이 플래쉬 메모리에는 크게 NOR 형과 NAND 형이 있으며 장단점은 아래와 같다.

NAND형은 프로그램/소거가 빠르지만 읽기, 쓰기가 바이트화가 불가능하고, 읽기 위해서 일반 메모리처럼 단순히 데이터버스와 Address버스를 사용하여 읽을 수 있

는 것이 아닌 소프트웨어 적인 작업을 해야 하므로 번거로울 뿐 아니라 이점 때문에 별도의 실행코드 저장용 롬을 필요로 한다. 용도로는 대용량 데이터 저장형으로 쓰인다.

NOR형은 읽기, 쓰기가 바이트화(Random Access가능)가 가능하지만 쓰기/지우기가 느리다는 단점이 있다. 용도로는 코드저장용으로 쓰여서 직접 실행이 가능하다.

NOR형의 경우는 인텔과 AMD사등에서 생산되고 있다. NAND형은 도시바, 삼성등에서 생산하고 있다.

우리는 여기서 NOR형 플래쉬에 대하여 알아 볼 것이며, 스트라타 플래시에 대하여 직접 프로그램까지 해보기로 하자.

4.1.1. NOR 형 플래쉬

NOR형 플래쉬 메모리는 일반 메모리를 읽는 것처럼 데이터를 읽을 수가 있지만, NAND형 플래쉬 메모리는 단지 Address라인과 Data라인의 접근으로 값을 읽을 수 없고, 소프트웨어적으로 명령/데이터주소를 쓰고, ECC를 체크하는등의 복잡한 과정을 거쳐야 한다고 했다. NOR형 메모리는 이런 장점을 가지고 있는 반면에 NAND형보다 만들기가 어렵기 때문에 가격이 비싸다.

인텔의 플래시 메모리가 많이 쓰이는데 인텔제품은 크게 스트라타 플래시와 부트블럭 플래시 이렇게 두 종류로 나뉜다.

■ 스트라타 플래시/부트블럭 플래시

스트라타 플래시와 부트블럭 플래시메모리의 차이점을 보면, 우선 쓰기를 할 때 부트블럭 플래시 메모리는 한번에 한개의 워드만을 쓸 수 있는 반면 스트라타 플래시 메모리는 한번에 16개의 워드를 쓸 수 있는 장점으로 전체적으로 데이터를 쓰는 시간을 줄일 수 있다. 물론 한개의 워드씩 쓰는 것도 가능하다.

지울때는 블록 사이즈단위로 지워지게 되는데 부트블럭 플래쉬 메모리의 경우 TOP와 BOTTOM형이 존재하고 BOTTOM형의 경우 8K 데이터 블록들 + 64KByte 데이터 블록들의 구조로, TOP형의 경우 64KByte 데이터 블록들 + 8KByte 데이터 블록들들로 구성된다.

스트라타 플래쉬 메모리의 경우에는 128KByte 의 지우기 블록사이즈를 갖는다.

이 블록의 크기차이점은 큰것과 작은 것이 모두 각각 장단점이 있으며 많은 양의 데이터를 쓴다면 블록이 큰쪽이 작업시간이 빠르지만 작은 양의 데이터를 자주 쓴다면 부트블럭이 작은 쪽이 유리하다.

용량면에서는 부트블럭 플래시는 1MByte, 2MByte, 4MByte가 존재하며 스트라타 플래시는 32Mbit(예,28F320J3A), 64Mbit(예,28F640J3A), 128Mbit(예, 28F128J3A) 등이 있다.

4.2. NOR형 인텔 스트라타(28F128J3A) 접근하기

RC28F128J3A를 예로 들어 이름을 보면 처음 RC는 패키지 타입을 나타내는데 RC인 경우는 64-Ball Easy BGA타입을 E는 56 Lead TSOP타입을 GE는 48-Ball VF BGA타입을 나타낸다. 28F는 인텔의 모든 플래쉬 제품에 붙는 문자이며, 그 다음 128은 용량을 나타낸다. 그 다음 J는 Intel Strata Flash memory임을 나타내고, 3은 Vcc/Vpen이 3V동작임을 나타낸다.

이 NOR형 플래쉬는 X8 모드와 X16 모드가 존재하는데 이는 BYTE핀에 LOW를 입력하면 X8 모드로 D8-D15 핀은 사용되지 않으며, X16모드의 경우 A0번은 사용하지 않는다. X8모드의 경우 128KByte의 지우기 블록크기를 가지며, X16의 경우는 64KByte의 지우기 영역을 갖는다. 플래시 메모리 내의 데이터 내용을 읽을 때는 일반 메모리 읽듯이 읽으면 되지만, 플래시 메모리에 값을 써넣기 위해서, 또는 플래쉬 정보를 읽기 위해서는 플래시 메모리에 해당 명령어를 써서 데이터를 작업 영역을 바꾸어 주어야 한다. 명령어 셋에 대한 28FXXXJ3A 메뉴얼을 보면

Command	Scalable or Basic Command Set ⁽²⁾	Bus Cycles Req'd.	Notes	First Bus Cycle			Second Bus Cycle		
				Oper ⁽³⁾	Addr ⁽⁴⁾	Data ^(5,6)	Oper ⁽³⁾	Addr ⁽⁴⁾	Data ^(5,6)
Read Array	SCS/BCS	1		Write	X	FFH			
Read Identifier Codes	SCS/BCS	≥ 2	7	Write	X	90H	Read	IA	ID
Read Query	SCS	≥ 2		Write	X	98H	Read	QA	QD
Read Status Register	SCS/BCS	2	8	Write	X	70H	Read	X	SRD
Clear Status Register	SCS/BCS	1		Write	X	50H			
Write to Buffer	SCS/BCS	> 2	9, 10, 11	Write	BA	E8H	Write	BA	N
Word/Byte Program	SCS/BCS	2	12, 13	Write	X	40H or 10H	Write	PA	PD
Block Erase	SCS/BCS	2	11, 12	Write	BA	20H	Write	BA	D0H
Block Erase, Program Suspend	SCS/BCS	1	12, 14	Write	X	B0H			
Block Erase, Program Resume	SCS/BCS	1	12	Write	X	D0H			
Configuration	SCS	2		Write	X	B8H	Write	X	CC
Set Block Lock-Bit	SCS	2		Write	X	60H	Write	BA	01H
Clear Block Lock-Bits	SCS	2	15	Write	X	60H	Write	X	D0H
Protection Program		2		Write	X	C0H	Write	PA	PD

와 같이 나온다.

여기서 Addr항목에서 X는 플래쉬 영역이면 어디라도 상관 없음을 나타내고, BA는 블록의 주소를, PA는 쓰려는 개별 주소를 표시한다.

각 명령어 들에 대한 적용은 아래 예를 통해 하나하나 보도록 할 것이며, 28FXXXJ3A 메뉴얼에 보면, 각 작업에 대한 순서도들이 나오므로 이를 참조하며, KelbJTag의 소스를 참조하도록 하라.

4.2.1. 플래시 ID 읽기 (Read Identifier Codes)

아래에서 우리는 이 정보중에 플래쉬 ID를 읽는 작업을 볼 것이며, 이 플래시의 ID값을 확인하는 절차를 통해 우리는 각 플래시의 타입을 알 수 있으며, 그에 따라 프로그램 방식을 바꿀 수 있을 것이다. 이 ID에는 제조회사번호, 디바이스코드 등의 정보를 포함하고 있으며, 메뉴얼에 보면 플래시에 대한 명령어와 ID에 대한 자세한 정보가 나온다. 우선 **Read Identifier Codes** 명령어는 위의 테이블을 보면 90H라는 것을 알 수 있다. 읽는 데이터는 **Manufacture ID**는 Offset 00H번지에 **Device Code**는 Offset 01H 번지에 존재하게 되는데 이 디바이스의 경우 **Manufacture Code**는 89H이며, **Device Code**는 32-Mbit의 경우에는 16H, 64Mbit은 17H, 128Mbit은 18H이다.

ID 레지스터의 구성은 아래와 같다.

Code		Address ⁽¹⁾	Data
Manufacture Code		00000	(00):89
Device Code	32-Mbit	00001	(00):16
	64-Mbit	00001	(00):17
	128-Mbit	00001	(00):18
Block Lock Configuration		x0002 ⁽²⁾	
• Block Is Unlocked			D0[0]
• Block Is Locked			D0[1]
• Reserved for Future Use			D[7:1]

이 시스템은 2개의 플래쉬메모리를 사용해 32비트 데이터너비를 사용하므로 0017 0017H와 같은 형식으로 표시될 것이며, 명령을 줄 경우에도 0090 0090H코드를 주어야 한다. 이 후 작업에서는 이해를 위하여 하나의 플래쉬접근 만을 다룬다.

1. 플래쉬 영역에 ID읽기 명령인 0x90을 쓴다.

플래시에 **Read_ID(0x90)** 명령을 써 주어 플래쉬의 읽을 영역을 **Identifier Code**영역으로 변경한다. 이때 접근하는 주소는 플래시 영역내에만 존재하면 되고, 그 주소에 **Read_ID(0x90)**의 값을 쓰면된다.

```
*(FLASH_BASE)=0x90;
```



2. 해당 메모리에서 값을 읽어온다.

```
/* 플래쉬의 베이스주소+00H에서 아이디를 읽는다. */
```

```
Manufacture_ID=*(FLASH_BASE+0x00);
```

```
/* 메뉴얼 상에 나온 Offset 주소 1은 워드이기 때문에 2를 곱해주도록 한다. 사실 2개의 플래쉬를 쓸 경우에는 4를 곱해주어야 한다.*/
```

```
Device_ID=*(FLASH_BASE+2*0x01);
```



3. Read Array명령(0xFF)를 주어 일반 읽기 모드로 변경한다.

```
*(FLASH_BASE)=0xFF;
```

4.2.2. Lock 정보 읽기/세팅

플래쉬에 어떠한 값을 쓰기 위해서는 쓰기금지 락이 걸려있는지 확인하고, 걸려있으면 해제한 후 써야 한다. 락이 걸려있는 지를 확인 하는 방법은 [4.2.1. ID레지스터]에서 보이듯이 오프셋 2H주소에 정보가 있다. 따라서, Read ID Register 명령(90H)를 플래쉬 영역에 써 준 다음 Base 주소의 오프셋 2H에서 읽어 오면 된다. 이 값의 비트[0]의 값이 0이면 그 블록은 unlock 상태이며, 1이면 Lock상태이다.

락을 세팅 또는 클리어하기 위해서는 Block Lock 명령인 60H을 플래쉬 영역에 써 준 다음 블록의 Base주소에 락을 걸때는 01H를 써주고, Unlock 시킬때는 D0H를 써 주면 된다. 이 부분에 대해서는 다음 플래시 블록지우기 및 플래시 영역에 데이터 쓰기를 통해 볼 것이다.

4.2.3. Status Register (Read Status Register) 상태 확인

상태를 확인하기 위한 명령은 70H이며, 이 레지스터의 정보에는 비트 7번은 WRITE STATE MACHINE STATUS비트로 1이면 Ready 상태, 0이면 Busy 상태이다. 비트6은 지우기 작업이 완료되었는지를, 비트 5는 지우기 또는 Unlock시에 성공을 했는지, 비트 4는 프로그램(쓰기) 와 Lock을 셋하는 작업이 성공했는지 비트 3은 쓰기 전압상태에 대하여 비트 2는 프로그램(쓰기)이 완료되었는지 비트 1은 디바이스가 락이 걸려있는지를 나타내고 비트 0번은 예약비트이다. Block Erase, Set Lock Bit, Clear Lock Bit작업을 한 후에는 플래시의 영역에서 별도의 명령(70H)을 주지 않고 Status Register를 바로 읽어 들여 작업이 수행중인지 끝났는 지 알 수 있다. 이 부분 역시 아래 플래시 블록지우기 및 플래시 영역에 데이터 쓰기에서 볼 것이다.

4.2.4. 플래시 블록 지우기 (Block Erase)

블록지우기 명령은 20H이며 이 명령코드를 플래시에 써 주어야 하며, 써 준 다음에는 Block Erase/Program Resume 명령(0xD0)을 줌으로써 데이터를 고정시켜야 한다. 이 Block Erase/Program Resume 명령은 쓰기에서도 사용된다. 이 플래쉬메모리에 데이터를 고정시키는 작업을 보통 퓨징작업이라 한다. 지우기 작업이 끝나면 Status Register를 읽어 지우기가 성공했는지 반드시 확인해야 한다.

작업순서를 보면 우선 플래시에 쓰기금지 락이 걸려 있는지 확인하고, 락이 걸려 있다면 락을 풀어주는 작업을 미리한 후 블록을 지워준 후 마지막에서 다시 락을 걸어주어 데이터가 함부로 지워지는 것을 막도록 한다.

1. 블록에 락이 걸려있는지 확인 후 락이 걸려있다면 락을 푼다.


```

/* 0x90명령으로 오프셋(2H)주소에서 값을 읽어와 비트 0번을 체크하여 락이 걸려있는지
확인한다.*/
*(FLASE_BASE)=0x90;
/* 해당 블록의 오프셋 2에서 데이터를 읽어온다. */
Data=*(FLASH_BASE+BLOCK+2*2);
/*락이 걸려 있다면 Clear Lock명령(0x60)을 주고 0xD0의 값을 써준다.*/
if( Data & 0x01 ){
    *(FLASH_BASE)=0x60;
    *(FLASH_BASE+BLOCK) = 0xD0;
}

```



2. Status 레지스터의 값을 확인하여 위의 작업이 끝났는지 확인한다.

Status 레지스터의 비트 7번을 확인함으로써 위의 명령이 수행을 끝났는지 확인하여 작업이 끝날때 까지 루프를 반복한다. 이 확인 작업은 Status 레지스터의 비트 7번이 1로 될때 까지 대기하면 된다.

```
while (!(*(FLASH_BASE)& 0x80 );
```



3. 해당 블록 주소에 Erase Block명령(0x20)을 주어 블록을 지운다.

```
*(FLASH_BASE+BLOCK)=0x20;
```



4. Erase/Program Resume 명령(0xD0)을 주어 데이터를 고정한다.

```
*(FLASH_BASE+BLOCK) = 0xD0;
```



5. 2번항과 같이 명령 수행이 끝났는지 확인한다.

```
while (!(*(FLASH_BASE)& 0x80 );
```



6. Status Register를 읽어서 위의 작업이 성공했는지 확인한다.

여기서는 비트 3을 읽어서 프로그램(쓰기) 전압에 이상이 있었는지를 비트 5을 읽어서 지우기 작업이 성공적으로 행해졌는지 확인해야 한다

```

/* Status Register를 읽어 온다. */
STR = *(FLASH_BASE)
/* 비트 3이나 비트 5번이 만약 1이라면 지우기 실패한것이다 */
if ( (STR&0x08) || (STR&0x20) )          /*실패*/
    Rtn=FAIL;
else                                     /*성공*/
    Rtn=SUCCEED;

```



7. Clear Status 명령 (0x50)을 주어 Status Register를 클리어 한다.

```
*(FLASH_BASE) = 0x50;
```

8. Read Array 명령(0xFF)을 주어 일반 읽기 모드로 변경한다.(반드시)
`*(FLASH_BASE) = 0xFF;`

4.2.5. 1바이트 또는 1워드 쓰기

쓰기 작업은 어떻게 보면 지우기 작업과 비슷하다. Word/Byte 프로그램(쓰기) 명령은 40H 또는 10H이며 이 명령을 쓰려는 플래쉬 주소에 써 준 다음 Block Erase /Program Resume 명령(0xD0)을 주어 데이터를 고정시킨다. 이렇게 작업을 한 후에는 Status Register를 읽어 쓰기가 성공했는지 반드시 확인해야 한다.

작업순서를 보면 우선 플래시에 쓰기금지 락이 걸려 있는지 확인하고, 락이 걸려 있다면 락을 풀어주는 작업을 한 후(여기서는 위에서 보았으므로 이 과정을 생략했다) 데이터를 쓰고 마지막에 다시 락을 걸어주어 데이터가 함부로 지워지는 것을 막도록 한다.

1. 쓰기 명령어(0x40 or 0x10)을 준다.
`*(FLASH_BASE) = 0x40;`

2. 쓰려는 주소에 쓰려는 값을 쓴다.
`*(FLASH_BASE+Addr) = Data;`

3. 명령 수행이 끝났는지 확인한다.
`while (!(*FLASH_BASE)& 0x80);`

4. Erase/Program Resume 명령(0xD0)을 주어 데이터를 고정한다.
`*(FLASH_BASE+BLOCK) = 0xD0;`

5. 3번항과 같이 명령 수행이 끝났는지 확인한다.
`while (!(*FLASH_BASE)& 0x80);`

6. Status Register를 읽어서 위의 작업이 성공했는지 확인한다.
 여기서는 비트 3을 읽어서 프로그램(쓰기) 전압에 이상이 있었는지를 비트 4을 읽어서 쓰기 작업이 성공적으로 행해졌는지 확인해야 한다.
`/* Status Register를 읽어온다. */
 STR = *(FLASH_BASE);
 if ((STR&0x08)|| (STR&0x10)) /*실패*/`

```

else
    Rtn=FAIL;
    Rtn=SUCCEED;

```



7. Clear Status 명령 (0x50)을 주어 Status Register를 클리어 한다.

```

*(FLASH_BASE) = 0x50;

```



8. Read Array명령(0xFF)를 주어 일반 읽기 모드로 변경한다.

```

*(FLASH_BASE) = 0xFF;

```

4.2.6. 버퍼 쓰기(Write To Buffer)

이 명령은 블록 플래시에서는 지원하지 않으며 스트라타에서만 지원한다. 명령코드는 E8H명령값을 가지며 이 명령을 쓰려는 블록의 주소에 써 주면 되고, 쓸 수 있는 최대 데이터의 양은 버퍼의 크기(32Byte)만큼이다. 여기서 버퍼쓰기를 하기 위해서 eXtended Status Register(XSR)의 비트 7번이 1을 가지고 있어야 쓰기버퍼가 가능한 상태이다. 이 XSR 레지스터는 쓰려는 플래쉬 블록주소에서 읽어온다. 나머지 작업은 1바이트 쓰기 방식과 똑같다.

1. 버퍼쓰기 명령어(0xE8)을 준다.

```

*(FLASH_BASE+BA) = 0xE8;

```



2. 명령 수행이 끝났는지 확인하기 위하여 XSR의 7번 비트가 1인지를 확인한다.

```

while (!(* (FLASH_BASE+BA)& 0x80 ));

```



3. 버퍼에 쓰려는 워드수를 써준다.
여기서 쓰려는 워드길이는 0부터 시작하므로 0이란 값을 주면 1워드 1을 주면 2워드를 뜻한다.

```

*(FLASH_BASE+BA) = WRITE_LEN;

```



4. 쓰려는 데이터를 쓰려는 주소에 써 준다.
아래에서 쓰기버퍼의 최대크기로 WRITE_LEN=16으로 두었다.

```

for (u8Index=0; u8Index<WRITE_LEN; u8Index++) {
    *(FLASH_BASE+PA) = _Buffer[u8Index];
    PA += 2;
}

```



5. Erase/Program Resume 명령(0xD0)을 주어 데이터를 고정한다.

```
*(FLASH_BASE+BLOCK) = 0xD0;
```



6. 명령 수행이 끝났는지 확인한다.

```
while (!(*(FLASH_BASE)& 0x80 ));
```



7. Status Register를 읽어서 위의 작업이 성공했는지 확인한다.

여기서는 비트 3을 읽어서 프로그램(쓰기) 전압에 이상이 있었는지를 비트 4을 읽어서 쓰기 작업이 성공적으로 행해졌는지 확인해야 한다.

```
/* Status Register를 읽어온다. */
STR = *(FLASH_BASE);
if ( (STR&0x08)||((STR&0x10) ) /*실패*/
      Rtn=FAIL;
else
      Rtn=SUCCEED;
```



8. Clear Status 명령 (0x50)을 주어 Status Register를 클리어 한다.

```
*(FLASH_BASE) = 0x50;
```



9. Read Array명령(0xFF)을 주어 일반 읽기 모드로 변경한다.

```
*(FLASH_BASE) = 0xFF;
```

여기서 KELB 하드웨어 구성이 2개의 인텔 스트라타 플래시가 2뱅크로 구성된 32비트 구조로 되어 있다고 했다. 따라서, 한 명령어(0x11)을 주고자 한다면, 두개의 플래시 모두 주어야 하므로 0x00110011을 주어야 할 것이다.

5. KELBJTAG 사용하기

여기서 위에서 행한 규칙들을 따라서 프로그램한 [KELBJTAG]의 사용법과 구조를 보인다. 이 JTAG 프로그램은 상용으로 만든 것이 아니기 때문에 버그가 있을 수 있으며, 단지 해석차원에서 보았으면 한다. 따라서, 수정/배포는 제약이 없으며, 16비트 스트라타 플래시 2개를 사용한 32비트 버스에 대한 코딩이다.

5.1. KELBJTAG 사용법

KELBJTAG는 다음과 같은 메뉴형식으로 구성되어 있다. KelbJTag를 실행시켜보자.

```
[root@Dci jtag]# ./MyJtag
Open LPT Port : 378
-----<< MAIN MENU >>-----
[J] -- Move to [J]Tag Menu.
[F] -- Move to [F]lash Menu.
[H] -- [H]elp.
[E] -- [E]xit.
# Command : █
```

위와 같은 메인메뉴가 나온다.

5.1.1. JTAG 메뉴

JTag Menu로 가기 위해 메인 메뉴에서 'J'를 치면

```
-----<< JTAG MENU >>-----
[D] -- [D]etect Pxa255 JTag ID.
[P] -- [P]rint Cells Number with High Value.
[R] -- [R]ead Cells State from Pxa255.
[W] -- [W]rite Cells State.
[H] -- [H]elp.
[M] -- [M]ove to Main Menu.
# Command : █
```

위와 같이 JTAG MENU가 나오는데 각 메뉴가 하는일은 아래와 같다.

키보드	명령내용
D	보드의 JTAG ID를 읽어 PXA255 ID와 비교한다.
P	버퍼내의 HIGH값을 가지고 있는 셀들의 번호를 표시한다.
R	보드에서 버퍼로 셀들의 값을 읽어 온다.
W	특정핀에 출력을 주기 위한 메뉴가 나온다.
H	도움말 기능으로 위와 같은 메뉴가 나온다.
M	상위 즉, Main Menu로 나간다.

이중에서 'W' 명령에 대해서 보자.

```
-----<< WRITE JTAG MENU >>-----
[P] -- [P]rint Cells Number with High Value.
[W] -- [W]rite Cell's Value.
[C] -- Force to [C]lear All Cells' Value.
[U] -- [U]pdate Cell with Current Setting Cells'Info, Not Reading Cells State.
[H] -- [H]elp.
[M] -- [M]ove to Jtag menu.
# Command : █
```

키보드	명령내용
-----	------

P	버퍼내의 HIGH값을 가지고 있는 셀들의 번호를 표시한다.
W	특정 출력을 주고 싶은 핀번호와 출력값을 세팅할 수 있다. 이렇게 세팅을 하고 [U] 명령을 주어야 할당된다.
C	모든 핀들을 0상태로 만든다. 이렇게 세팅을 하고 Update Pin State 명령을 주어야 할당된다.
U	버퍼의 셀들을 값을 설정한 후에는 이 [U]명령으로 업데이트 시켜주어야 핀들에 설정한 값들이 설정된다.
H	위의 메뉴를 나타낸다.
M	상위 메뉴, 즉 JTAG Menu로 나간다.

여기서 잠깐 특정 핀에 특정값을 내보내는 작업을 해보자. Pin번호 23번인 GPIO0_OUT에 1의 값을 출력한 것이다. 이에 앞서 GPIO핀 0번을 Output으로 정의하기 위하여 Pin 22번인 GPIO0_EN을 1로 만들어야 하며 이런 설정 작업이 끝난 후에는 'U' 명령을 주어야 실제 보드에 적용된다.

```

-----<< WRITE JTAG MENU >>-----
[P] -- [P]rint Cells Number with High Value.
[W] -- [W]rite Cell's Value.
[C] -- Force to [C]lear All Cells' Value.
[U] -- [U]pdate Cell with Current Setting Cells'Info, Not Reading Cells State.
[H] -- [H]elp.
[M] -- [M]ove to Jtag menu.
-----
# Command : w
[W]rite Cell's Value

----- Write Cell Number[Dec] : 23
----- Write Value[1:0] : 1
Cell Number : 23, Value : 1
Is it right?(Y/N)y
Set Cell State.... You should Update Cell to apply this cell's value!

.... Please, Click Enter Key ....

```

이상이 JTAG에 대한 메뉴 내용이며, 아래는 FLASH MENU에 대한 설명이다.

5.1.2. Flash Menu

Flash 메뉴는 플래쉬에 데이터를 써 넣기 위한 메뉴이고 이곳으로 가기위하여 메인메뉴에서 F를 치면 아래와 같은 메뉴가 생성된다.

```

-----<< Flash MENU >>-----
[D] -- [D]etect Flash .
[W] -- [W]rite Memory Menu.
[H] -- [H]elp.
[M] -- [M]ove to main menu.
-----
# Command : 

```

키보드	명령내용
D	플래쉬 ID를 읽어 인텔 스트라타플래쉬 아이디와 비교한다.
W	플래쉬 영역에 데이터를 써 넣는 메뉴로 읽어들여 메모리에 쓸 파일이름과 쓰기 시작할 주소, 길이를 적는 항목들이 나온다. 이때 길이를 0으로 주면 파일의 길이만큼 쓰게 되어 있다.
H	위의 메뉴 설명이 나온다.
M	상위 메뉴 즉, Main Menu로 나간다.

그러면 플래쉬에 부트로더를 쓰는 작업을 위의 메뉴에서 보도록 하자. 0x0000번 지부터 부트로더 파일 크기만큼 쓰면 된다. 부트로더의 이름은 Kelb_Boot이다.

```

-----<< Flash MENU >>-----
[D] -- [D]etect Flash .
[W] -- [W]rite Memory Menu.
[H] -- [H]elp.
[M] -- [M]ove to main menu.
-----
# Command : w
[W]rite Memory Menu

- File Name : blob
- Write Start Address[Hex] : 0
- Write Length[Dec] (If Length=0, Length=File Size) : 0
[ blob ] File Size : 39756 Byte
Erase Buffer: |=====| 100%
Write Buffer:  ==\      4%

```

5.2. KELB JTAG 소스구조

이상에서 우리는 KelbJTag의 사용법을 알아 보았다. 아래에 JTag 소스구조에 대하여 보도록 하자. 저자가 작성한 것이라 소스의 난이도 및 버그에 대해서는 자신이 없지만, 이 소스를 통하여 보드를 디버깅을 하였고 플래쉬에 부트로더를 써 넣는데 사용되었으므로 틀림없이 동작되는 코드이다.

파일이름	소스내용
Kelb.c	이 파일에는 Main Menu에 대한 처리가 되어있다.
JTag.c	이 파일에는 JTag Menu에 대한 처리가 들어 있으며, 틀린 케이블을 사용하고 있다면 이 파일의 맨위에서 #define처리한 TDI, TDO, TCK를 알맞게 수정하여 주면 될 것이다.
PrnPrt.c	실제 프린터포트로 시그널을 내보내기 위한 루틴들이 들어 있다.
Pxa255.c	Pxa255의 셀들에 접근하여 특정 번지의 주소로의 접근등을 담당하는 함수들로 구성된다.

Flash.c 28F128J3A.c	이 파일에는 Flash Menu 에 대한 처리가 되어 있다. 헤더파일인 Flash.h 에는 인텔 스트라타 플래쉬의 명령어들이 정의되어 있다.
util.c	이 파일에는 바 그래프를 표시하는 함수인 UpdateProgress() 함수가 정의된다.

SoftWare

부팅에서 프람프트까지

4

부팅과정요약

이장 이후로 ARM 리눅스의 부팅과정에 대하여 소스를 분석하고, 수정하여 사용해야 할 것이 어디인지를 볼 것이다. 이 장에서는 큰 블록을 잡아 부팅과정의 소스를 보기전에 부팅과정의 요약을 통하여 부팅이 어떻게 진행이 되는지를 큰 블록을 잡아 보도록 한다.

이렇게 함으로써 앞으로 진행을 해나가는 데 소스에만 빠져 맥을 놓치지 않고 맥락을 따라 갈 수 있을 것이다. 사실 리눅스의 소스를 본다는 것은 끝이 없는 일이 될 수 있으며, 한번 빠지면 큰 맥은 잊어버린채 소스를 따라가는데 정신을 빼앗기게 되어 밤을 세기가 다반이 될 것이다. 이런것도 물론 재미있지만, 맥을 잡아놓고 따라가는 것이 성취에 더 많은 도움이 되리라 믿는다.

아래 리눅스 부팅과정에 대한 요약도를 보인다. 이는 블록을 잡아 간략한 설명만으로 마치고 해당장에서 소스를 보면서 좀더 상세히 보도록 하자.

<아직 작업하지 않았음.... 죄송>

5

부트로더

부트로더는 시스템의 전원은 온하는 순간 가장 먼저 시작하며 이것이 하는 역할은 리눅스 커널의 시작을 위하여, **CPU**의 동작속도, 메모리설정, 기본 하드웨어 초기화를 하고, 리눅스를 램 메모리로 로드한 후 리눅스로의 점프를 한다.

1. BLOB 설치하기

부트로더는 보드의 초기화와 저장 매체 또는 통신을 통해 커널을 읽어서 메모리의 적절한 위치로 적재한 다음, 커널로 제어를 옮기는 역할을 수행한다. 또한 개발 중인 커널의 경우에 네트워크를 통한 커널 image의 다운로드도 제공한다.

여기서는 BLOB을 사용하여 부트로더를 만들어 볼것인데, 이는 BLOB이 부트로더를 이해하는데 가장 쉽고 검증된 코드라고 필자는 생각하기 때문이다.

홈페이지는 <http://www.lart.tudelft.nl/> 이며 이곳에서 다운로드 받을 수 있지만, 이 글을 집필하는 현재 SA1110까지의 지원만 가능하고, XScale에 대한 지원은 하지 않으므로, BLOB를 수정하여 XScale 을 지원하도록 코딩된 <ftp://ftp.arm.linux.org.uk/pub/linux/arm/people/xscale/lubbock/06-02-2003/src/blob-lubbock-052903.tgz> 파일을 다운로드 받도록 한다.

이하 진행사항은 README 파일을 참조한 것이다. 이 BLOB을 컴파일하기 위해서는 커널소스가 필요하므로(왜 필요한지는 이 후 보인다) 우선 BLOB 컴파일시 필요한 리눅스 커널을 설치하는 것을 먼저 보도록 한다.

1.1. XScale용 리눅스 커널 설치하기

BLOB설치과정에서 보게 되겠지만, 이 BLOB를 설치하고, 컴파일 하기위해서는 우선 ARM용 리눅스 커널이 설치 되어 있어야 한다. 이유는 소스를 보면서 보게 될 것이며, 여기서 설치되는 커널코드를 수정하여 커널 포팅까지 진행하도록 할 것이므로, 여기서 리눅스 커널설치를 착오가 없도록 하자.

☞ 필요파일

종류	다운로드 파일명
리눅스커널	ftp://ftp.kr.kernel.org/pub/linux/kernel/v2.4/linux-2.4.19.tar.gz
ARM 패치	ftp://ftp.arm.uk.linux.org/pub/linux/arm/kernel/v2.4/patch-2.4.19-rmk7.gz
XScale 패치	ftp://ftp.arm.uk.linux.org/pub/linux/arm/people/nico/diff-2.4.19-rmk7-pxa1.gz

우리는 2.4.19에 대한 커널에 대한 작업을 하도록 할 것이며 작업 디렉토리는 /usr/src/Kernel/ 디렉토리로 하겠다. 아래의 과정은 리눅스 커널 소스를 풀고, 소스 트리의 최상위 디렉토리에서 ARM 패치와 XScale 패치를 실행한다.

```
[root@Dci kernel]# ls
```

```
diff-2.4.19-rmk7-pxa1.gz linux-2.4.19.tar.gz patch-2.4.19-rmk7.gz
[root@Dci kernel]# tar -zxvf linux-2.4.19.tar.gz
[root@Dci kernel]# ls
diff-2.4.19-rmk7-pxa1.gz linux linux-2.4.19.tar.gz patch-2.4.19-rmk7.gz
[root@Dci kernel]# cd linux2.4.19
[root@Dci linux2.4.19]# gzip -cd ../patch-2.4.19-rmk7.gz | patch -p1
[root@Dci linux2.4.19]# gzip -cd ../diff-2.4.19-rmk7-pxa1.gz | patch -p1
[root@Dci linux2.4.19]# ls
COPYING Documentation Makefile REPORTING-BUGS arch fs init kernel
mm scripts
CREDITS MAINTAINERS README Rules.make drivers include ipc lib
net
```

1.2. BLOB 설치하기

이상 커널소스를 설치하였다면 BLOB의 압축을 풀고, 이상없이 모든것이 설치되었는가를 알아보기 위하여 lubbock 보드의 옵션을 통하여 컴파일 까지 진행하도록 한다. 이 후 우리가 정한 보드이름을 가지고 다시 컴파일하여 우리의 시스템에 적용하는 과정을 가질 것이다.

① 압축풀기

우선 /usr/tmp 디렉토리로 받은 파일을 복사한 후 압축을 푼다.

```
[root@Dci src]# tar -zxvf blob-lubbock-052903.tgz
[root@Dci src]# cd blob-xscale/
[root@Dci blob-xscale]# ls
AUTHORS ChangeLog Makefile.am NEWS RELEASE-NOTES config-pxa
configure.in include tools
COPYING INSTALL Makefile.in README aclocal.m4 configure doc
src utils
[root@Dci blob-xscale]#
```

② tools/rebuild-gcc를 실행한다.

(README 화일에는 두번실행하라고 되어 있다)

tools/rebuild-gcc를 \$(BLOBTOP)/ 디렉토리에서 실행하자. 이 과정으로 \$(BLOBTOP)디렉토리에 configure.in화일로 부터 configure실행화일이 만들어진다.

```
[root@Dci blob-xscale]# cp src/blob/acconfig.h ./
[root@Dci blob-xscale]# cp tools/rebuild-gcc ./
[root@Dci blob-xscale]# ./rebuild-gcc
$Id: rebuild-gcc,v 1.1.1.1 2001/06/27 19:47:42 erikm Exp $
Setting up for use with GNU C/C++ compilers
[root@Dci blob-xscale]# ls configure
configure
```

③ 크로스컴파일러 툴체인을 등록한다.

☞ Bash 의 경우

```
[root@Developers blob-2.0.5-pre2]# export CC=/usr/local/arm/arm-linux-gcc
```

```
[root@Developers blob-2.0.5-pre2]# export OBJCOPY=/usr/local/arm/arm-linux-objcopy
```

④ Configuring 작업

이 작업을 하기위해서는 리눅스 커널 소스가 설정되어 있어야 한다. PXA255타입의 타겟보드를 아무보드나 설정하기로 하자.

☞ 리눅스 커널 Configure

```
[root@Developers tmp]# cd linux
[root@Developers linux]# pwd
/usr/tmp/linux
[root@Developers linux]# make menuconfig
...시스템타입 PXA-base ARM system type으로, PXA Implementations메뉴에서 타겟보드를 아무것으로 설정한 후 저장하고 빠져나온다.
```

☞ BLOB Configure

"./configure --with-linux-prefix=/path/to/armlinux/source --with-board=boardname ar m-unknown-linux-gnu" 형식의 명령어로 --with-linux-prefix=[리눅스 소스 디렉토리]를 --with-board=[보드이름]을 적어준다.

```
[root@Dci blob-xscale]# ./configure --with-board=lubbock --with-cpu=pxa255 --with-linux-prefix=/usr/src/kernel/linux-2.4.19/ --with-eth=smc91x
Configuration
-----
Target cpu                pxa255
Target board              Intel Lubbock
C compiler                /usr/local/arm/bin/arm-linux-gcc
C flags                  -Os -I/usr/src/kernel/linux-2.4.19/include -Wall -march=armv4 -mtune=strongarm1100 -fomit-frame-pointer -fno-builtin -mapcs-32 -DCPU_pxa262
Linker flags              -static -nostdlib
Objcopy tool              /usr/local/arm/bin/arm-linux-objcopy
Objcopy flags             -O binary -R .note -R .comment -R .bss -S
Ethernet support          smc91x
USB support               no
Clock scaling support     no
Memory test support       no
Debugging commands support no
LCD support               no
MD5 support               no
Run-time debug information no
```

⑤ 컴파일 하기

컴파일은 make 명령만으로 한다.

```
[root@Dci blob-xscale]# make
```

에러없이 컴파일이 끝났다면, 바이너리 이미지가 생성했는가를 보도록 하자. 이들은 src/blob 디렉토리에 생성되며, 최종화일의 화일명은 blob이다.

```
[[root@Dci blob-xscale]# pwd
/usr/src/blob-xscale
[root@Dci blob-xscale]# ls src/blob/blob*
src/blob/blob          src/blob/blob-rest    src/blob/blob-start    src/blob/blob-start-
```

```
chain-elf32
src/blob/blob-chain  src/blob/blob-rest-elf32  src/blob/blob-start-chain  src/blob/blob-start-elf32
```

아래과정에서 보게 되겠지만, 위의 화일들의 내용을 잠시 보고 넘어가도록 하자. blob-start부분과 blob-rest부분으로 나뉘어 있는데 먼저 start부분이 실행이 되고, 이 start부분에서 rest부분을 램으로 옮겨 rest부분으로 분기하여 실행된다. 이런 과정때문에 start와 rest부분이 나뉘어 지며, 각 화일들에 대해서 보면 blob-rest, blob-start는 바이너리 화일이고 blob-start-elf32와 blob-rest-rest-elf32화일은 ELF 파일 포맷이다. 최종적으로 start부분과 rest부분이 묶여서 blob 화일이 만들어 지게되는 것이다.

2. 우리의 보드이름으로 **BLOB** 사용하기

여기서는 자신의 보드이름을 사용하기 위하여 (여기서는 KELB를 사용한다)수정되어야 할 사항, 기타 자신의 시스템에 맞게 추가되어야 할 화일의 소스 및 내용에 대해서 보도록 하자.

우리는 위에서 lubbock 보드화일명을 이용하여 작업을 했다. 하지만, 어디 사람마음이 그런가? 자신이 만든 보드에 자신이 정한 이름을 사용하고 싶을 것이다. 여기서 이를 위한 작업을 하며 \$(BLOBTOP)/doc/porting.txt화일에서 설명하고 있으므로 이를 참조하도록 한다.

① \$(BLOBTOP)/configure.in 화일의 보드이름을 넣기위하여 Kelb보드에 대한 설정을 아래와 같이 해넣는다.

\$(BLOBTOP)/configure.in

```

75 AC_ARG_WITH(board, [ --with-board=NAME      Name of the target board
76                               Valid names are:
77                               assabet      Intel Assabet
78                               kelb        Korean Embeded Linux Board
...
94 AC_MSG_CHECKING(target board)
95 case "$board_name" in
....
106     kelb)
107         board_name="Korean Embeded Linux Board"
108         AC_DEFINE(KELB)
109         BLOB_PLATFORM_OBJ="kelb.o"
110         AC_MSG_WARN([Please check assabet memory config in arch/kelb.h])
111         BLOB_FLASH_OBJS="intel32.o"
112         DIAG_PLATFORM_OBJ="kelb.o"
113         use_cpu="pxa255"
114         use_lcd="no"
115         ;;

```

② \$(BLOBTOP)/acconfig.h화일에 우리보드의 정보를 추가한다.

\$(BLOBTOP)/acconfig.h

```

/* Define for Kelb boards */
#undef KELB

```

③ \$(BLOBTOP)/include/blob/arch.h화일에 보드의 헤더화일을 추가한다.

\$(BLOBTOP)/include/blob/arch.h

```

#if defined ASSABET
# include <blob/arch/assabet.h>

```



```
#elif defined KELB
#include <blob/arch/kelb.h>
...
```

④ 다음과 같이 자신의 시스템에 맞도록 헤더화일을 만들고, include/blob/arch/kelb.h로 복사한다.

아래화일은 lubbock.h파일과 ez-board의 헤더화일을 수정하여 만든 것이며, 특히 ez-board의 부트로더중 헤더화일은 주석이 잘 되어있어 그대로 올린다. 부록 CD의 blob에 수정된 파일이 있으니 복사해서 사용해도 된다.

Include/arch/kelb.h

```
#ifndef BLOB_ARCH_KELB_H
#define BLOB_ARCH_KELB_H

#include "blob/pxa.h"

/*
*****
* Bootloader, Kernel, Ramdisk의 램 및 플래쉬메모리에서의 위치를 설정한다.
*****
*/
/* BLOB의 첫번째 스테이지에 의해 로드되는 위치 */
#define BLOB_ABS_BASE_ADDR (0xA0000000)

/* 램영역의 BLOB, KERNEL, PARAM, RAMDISK의 위치 */
#define BLOB_RAM_BASE (0xA0100000)
#define KERNEL_RAM_BASE (0xA0200000)
#define PARAM_RAM_BASE (0xA0180000)
#define RAMDISK_RAM_BASE (0xA1000000)

/* 플래쉬롬 영역의 BLOB, KERNEL, PARAM, RAMDISK의 위치
아래에서 PARAM영역은 사용하지 않는다.*/

/* 부트로더 플래쉬영역 0x00000000 - 0x00040000*/
#define BLOB_FLASH_BASE (0x00000000)
#define BLOB_FLASH_LEN 0x40000 //(256 * 1024)
/* PARAM 플래쉬영역 : 사용안함*/
#define PARAM_FLASH_BASE (BLOB_FLASH_BASE + BLOB_FLASH_LEN)
#define PARAM_FLASH_LEN (0)
/* 커널 플래쉬영역 0x00040000 - 0x00140000*/
#define KERNEL_FLASH_BASE (PARAM_FLASH_BASE + PARAM_FLASH_LEN)

#define KERNEL_FLASH_LEN (1024 * 1024)
/* 램디스크 플래쉬영역 0x00140000 - 0x00540000*/
#define RAMDISK_FLASH_BASE (KERNEL_FLASH_BASE + KERNEL_FLASH_LEN)
#define RAMDISK_FLASH_LEN (4 * 1024 * 1024)

/* 커널 Boot Parameter 위치 : 여기서는 사용하지 않는다. */
#define BOOT_PARAMS (0xA0000100)

/* 램디스크의 압축이 풀렸을 때의 크기 */
```

```

#define RAMDISK_SIZE                (8 * 1024)

/*
*****
* XScale Pxa255의 내부 레지스터 상대 위치
*****
*/
#define MEMC_BASE                   0x48000000
#define SDRAM_BASE                   0xA0000000
#define OSCR_BASE                    0x40A00010
#define FPGA_REGS_BASE              0x08000000

#define MDCNFG_OFFSET               0x00
#define MDREFR_OFFSET               0x04
#define MSC0_OFFSET                  0x08
#define MSC1_OFFSET                  0x0C
#define MSC2_OFFSET                  0x10
#define MECR_OFFSET                  0x14
#define SXLCR_OFFSET                 0x18
#define SXCNFG_OFFSET               0x1C
#define FLYCNFG_OFFSET              0x20
#define SXMRS_OFFSET                 0x24
#define MCMEM0_OFFSET               0x28
#define MCMEM1_OFFSET               0x2C
#define MCATT0_OFFSET               0x30
#define MCATT1_OFFSET               0x34
#define MCIO0_OFFSET                 0x38
#define MCIO1_OFFSET                 0x3C
#define MDMRS_OFFSET                 0x40
#define BOOT_DEF_OFFSET             0x44

/*메모리 클럭 */
#define MEM_CLK                      100
#define SDRAM_CLK_DIV2               0

/*
*****
* 사용클럭에 대한 설정
*****
*/
/* Clock Enable Register 설정 */
#define CKEN_PWM0                    (0 << 0) // PWM0   Clock Enable
#define CKEN_PWM1                    (0 << 1) // PWM1   Clock Enable
#define CKEN_AC97                    (0 << 2) // AC97   Clock Enable
#define CKEN_SSP                     (0 << 3) // SSP    Clock Enable
#define CKEN_HWUART                  (0 << 4) // HWUART Clock Enable
#define CKEN_STUART                  (1 << 5) // STUART Clock Enable
#define CKEN_FFUART                  (1 << 6) // FFUART Clock Enable
#define CKEN_BTUART                  (1 << 7) // BTUART Clock Enable
#define CKEN_I2S                     (0 << 8) // I2S    Clock Enable
#define CKEN_NSSP                    (0 << 9) // NSSP   Clock Enable
#define CKEN_USB                     (0 << 11) // USB    Clock Enable
#define CKEN_MMC                     (0 << 12) // MMC    Clock Enable
#define CKEN_FICP                    (0 << 13) // FICP   Clock Enable
#define CKEN_I2C                     (0 << 14) // I2C    Clock Enable

```

```

#define CKEN_LCD                (0 <<16) // LCD      Clock Enable

#define CKEN_VAL                \
( CKEN_PWM0 | CKEN_PWM1 | CKEN_AC97 | CKEN_SSP | \
  CKEN_HUART | CKEN_STUART | CKEN_FFUART | CKEN_BTUART | \
  CKEN_I2S | CKEN_NSSP | CKEN_USB | CKEN_MMC | \
  CKEN_FICP | CKEN_I2C | CKEN_LCD )

/*
*****
* SDRAM에 대한 설정
*****
*/
/*
* MDCNFG 레지스터 설정
*/
/*1번 페어(0/1뱅크)에 대한 설정 */
#define MDCNFG_DE0              (1 <<0 ) // SDRAM 0   Partition Enable = 1
#define MDCNFG_DE1              (0 <<1 ) // SDRAM 1   Partition
#define MDCNFG_DWID0            (0 <<2 ) // SDRAM 0/1 BusWidth 32bit  = 0
#define MDCNFG_DCAC0            (1 <<3 ) // SDRAM 0/1 columne address count
#define MDCNFG_DRAC0_128        (1 <<5 ) // SDRAM 0/1 row address count
#define MDCNFG_DRAC0_256        (2 <<5 ) // SDRAM 0/1 row address count

#define MDCNFG_DNB0             (1 <<7 ) // SDRAM 0/1 bank count
#define MDCNFG_DTC0             (3 <<8 ) // SDRAM 0/1 Timing
#define MDCNFG_DADDR0           (0 <<10) // SDRAM 0/1 Address match
mode
#define MDCNFG_DLATCH0          (1 <<11) // SDRAM 0/1 must be set
#define MDCNFG_DSA1111_0        (1 <<12) // SDRAM 0/1 use sa1111 = 0
/*2번 페어(2/3뱅크)에 대한 설정 */
#define MDCNFG_DE2              (0 <<(0 +16))// SDRAM 2   Partition Enable = 1
#define MDCNFG_DE3              (0 <<(1 +16))// SDRAM 3   Partition
#define MDCNFG_DWID2            (0 <<(2 +16))// SDRAM 2/3 BusWidth 32bit  = 0
#define MDCNFG_DCAC2            (1 <<(3 +16))// SDRAM 2/3 columne address count
#define MDCNFG_DRAC2_128        (1 <<(5 +16))// SDRAM 2/3 row address count
#define MDCNFG_DRAC2_256        (2 <<(5 +16))// SDRAM 2/3 row address count

#define MDCNFG_DNB2             (1 <<(7 +16))// SDRAM 2/3 bank count
#define MDCNFG_DTC2             (3 <<(8 +16))// SDRAM 2/3 Timing
#define MDCNFG_DADDR2           (0 <<(10+16))// SDRAM 2/3 Address match
mode
#define MDCNFG_DLATCH2          (1 <<(11+16)) // SDRAM 2/3 must be set
#define MDCNFG_DSA1111_2        (1 <<(12+16)) // SDRAM 2/3 use sa1111 = 0

#define MDCNFG_STD_VAL          \
( MDCNFG_DE0 | MDCNFG_DE1 | MDCNFG_DWID0 | MDCNFG_DCAC0 \
  MDCNFG_DNB0 | MDCNFG_DTC0 | MDCNFG_DADDR0 | MDCNFG_DLATCH0 \
  MDCNFG_DSA1111_0 | MDCNFG_DE2 | MDCNFG_DE3 | MDCNFG_DWID2 | \
  MDCNFG_DCAC2 | MDCNFG_DNB2 | MDCNFG_DTC2 | MDCNFG_DADDR2 \
  MDCNFG_DLATCH2 | MDCNFG_DSA1111_2 )

#define MDCNFG_128_VAL          \
( MDCNFG_DRAC0_128 | MDCNFG_DRAC2_128 | MDCNFG_STD_VAL )
#define MDCNFG_256_VAL          \
( MDCNFG_DRAC0_256 | MDCNFG_DRAC2_256 | MDCNFG_STD_VAL )

```

```

/*최종 적용할 값 */
#define MDCNFG_VAL MDCNFG_256_VAL

/*
 * MDREFR 레지스터 설정
 */
#define MEM_CLK 100 //MEMCLK =100MHz로 설정
#define SDRAM_CLK_DIV2 0

#if MEM_CLK == 166
#define MDREFR_DRI 33 // MEM-Clock SDRAM refresh interval
#elif MEM_CLK == 133
#define MDREFR_DRI 29
#elif MEM_CLK == 100
#define MDREFR_DRI 24
#else
#error "PAX255의 속도를 설정하세요"
#endif

#define MDREFR_E0PIN (0 <<12) // SDCKE0 Enable
#define MDREFR_K0RUN (0 <<13) // SDCLK0 Enable
#define MDREFR_K0DB2 (0 <<14) // SDCLK0 divide-2

#define MDREFR_E1PIN (1 <<15) // SDCKE1 Enable
#define MDREFR_K1RUN (1 <<16) // SDCLK1 Enable
#define MDREFR_K1DB2 ((SDRAM_CLK_DIV2&1) <<17) // SDCLK1 divide-2

#define MDREFR_K2RUN (0 <<18) // SDCLK2 Enable
#define MDREFR_K2DB2 (0 <<19) // SDCLK2 divide-2

#define MDREFR_APD (0 <<20) // SDRAM/Static-Memory AutoPowerDown enable
#define MDREFR_SLFRSH (0 <<22) // Self Refresh (must be clear when reset)

#define MDREFR_K0FREE (0 <<23) // SDCLK0 Free
#define MDREFR_K1FREE (1 <<24) // SDCLK1 Free
#define MDREFR_K2FREE (0 <<25) // SDCLK2 Free
/*최종 적용할 값 */
#define MDREFR_VAL (MDREFR_DRI | MDREFR_SLFRSH \
| MDREFR_E0PIN | MDREFR_K0RUN | MDREFR_K0DB2 | MDREFR_K0FREE \
| MDREFR_E1PIN | MDREFR_K1RUN | MDREFR_K1DB2 | MDREFR_K1FREE \
| MDREFR_APD | MDREFR_K2RUN | MDREFR_K2DB2 | MDREFR_K2FREE )

/*
 * MRS 레지스터 설정
 */
// burst-length / burst-type / cas-latncy
#define MDMRS_P01_CMD_BURST_CFG (0x00 <<7 )
// burst-length / burst-type / cas-latncy
#define MDMRS_P23_CMD_BURST_CFG (0x00 <<16)
#define MDMRS_VAL \
(MDMRS_P01_CMD_BURST_CFG | MDMRS_P23_CMD_BURST_CFG)

/*
*****
 * Static 영역에 대한 설정

```

```

*****
*/
/*
* 스택뱅크 0,1 설정 (CS 0,1)
*/
// nCS0 Device Type 0:non-burst Flash 1:SRAM 4:latency I/O
#define MSC_CS0_RT (0 <<0)
// nCS0 Bus Width 0:32-bit 1:16-bit
#define MSC_CS0_RBW (0 <<3)
// nCS0 nOE/nWE assert 0 ~ 15 0~11 :10~120 nsec 12~15 :130/150/180/230 nsec
#define MSC_CS0_RDF (13 <<4)
// nCS0 next assert 0 ~ 15
#define MSC_CS0_RDN (2 <<8)
// nCS0 cs to cs period 0 ~ 7
#define MSC_CS0_RRR (2 <<12)
// nCS0 faster device = 1
#define MSC_CS0_RBUFF (0 <<15)

// nCS1 Device Type 0:non-burst Flash 1:SRAM 4:latency I/O
#define MSC_CS1_RT (0 <<(0 +16))
// nCS1 Bus Width 0:32-bit 1:16-bit
#define MSC_CS1_RBW (0 <<(3 +16))
// nCS1 nOE/nWE assert 0 ~ 15 0~11 :10~120 nsec 12~15 :130/150/180/230 nsec
#define MSC_CS1_RDF (4 <<(4 +16))
// nCS1 next assert 0 ~ 15
#define MSC_CS1_RDN (2 <<(8 +16))
// nCS1 cs to cs period 0 ~ 7
#define MSC_CS1_RRR (2 <<(12+16))
// nCS1 faster device = 1
#define MSC_CS1_RBUFF (0 <<(15+16))

#define MSC0_VAL ( MSC_CS0_RT | MSC_CS0_RBW | \
| MSC_CS0_RDF | MSC_CS0_RDN | MSC_CS0_RRR | MSC_CS0_RBUFF | \
| MSC_CS1_RT | MSC_CS1_RBW | \
| MSC_CS1_RDF | MSC_CS1_RDN | MSC_CS1_RRR | MSC_CS1_RBUFF )
/*
* 스택뱅크 2,3 설정 (CS 2,3)
*/
// nCS2 Device Type 0:non-burst Flash 1:SRAM 4:latency I/O
#define MSC_CS2_RT (0 <<0)
// nCS2 Bus Width 0:32-bit 1:16-bit
#define MSC_CS2_RBW (1 <<3)
// nCS2 nOE/nWE assert 0 ~ 15 0~11 :10~120 nsec 12~15 :130/150/180/230 nsec
#define MSC_CS2_RDF (15 <<4)
// nCS2 next assert 0 ~ 15
#define MSC_CS2_RDN (2 <<8)
// nCS2 cs to cs period 0 ~ 7
#define MSC_CS2_RRR (2 <<12)
// nCS2 faster device = 1
#define MSC_CS2_RBUFF (1 <<15)

// nCS3 Device Type 0:non-burst Flash 1:SRAM 4:latency I/O
#define MSC_CS3_RT (0 <<(0 +16))
// nCS3 Bus Width 0:32-bit 1:16-bit
#define MSC_CS3_RBW (1 <<(3 +16))
// nCS3 nOE/nWE assert 0 ~ 15 0~11 :10~120 nsec 12~15 :130/150/180/230 nsec
#define MSC_CS3_RDF (15 <<(4 +16))

```

```

// nCS3 next assert          0 ~ 15
#define MSC_CS3_RDN          ( 2 <<(8 +16))
// nCS3 cs to cs period 0 ~ 7
#define MSC_CS3_RRR          ( 2 <<(12+16))
// nCS3 faster device = 1
#define MSC_CS3_RBUFF        ( 0 <<(15+16))

#define MSC1_VAL ( MSC_CS2_RT | MSC_CS2_RBW \
                  | MSC_CS2_RDF | MSC_CS2_RDN | MSC_CS2_RRR | MSC_CS2_RBUFF \
                  | MSC_CS3_RT | MSC_CS3_RBW \
                  | MSC_CS3_RDF | MSC_CS3_RDN | MSC_CS3_RRR | MSC_CS3_RBUFF )

/*
* 스택뱅크 4,5 설정 (CS 4,5)
*/
// nCS4 Device Type          0:non-burst Flash  1:SRAM  4:latency I/O
#define MSC_CS4_RT            ( 0 <<0 )
// nCS4 Bus Width            0:32-bit  1:16-bit
#define MSC_CS4_RBW           ( 1 <<3 )
// nCS4 nOE/nWE assert 0 ~ 15 0~11 :10~120 nsec 12~15 :130/150/180/230 nsec
#define MSC_CS4_RDF           (15 <<4 )
// nCS4 next assert          0 ~ 15
#define MSC_CS4_RDN           ( 2 <<8 )
// nCS4 cs to cs period      0 ~ 7
#define MSC_CS4_RRR           ( 2 <<12)
// nCS4 faster device = 1
#define MSC_CS4_RBUFF         ( 0 <<15)

// nCS5 Device Type          0:non-burst Flash  1:SRAM  4:latency I/O
#define MSC_CS5_RT            ( 4 <<(0 +16))
// nCS5 Bus Width            0:32-bit  1:16-bit
#define MSC_CS5_RBW           ( 0 <<(3 +16))
// nCS5 nOE/nWE assert 0 ~ 15 0~11 :10~120 nsec 12~15 :130/150/180/230 nsec
#define MSC_CS5_RDF           (15 <<(4 +16))
// nCS5 next assert          0 ~ 15
#define MSC_CS5_RDN           ( 15 <<(8 +16))
// nCS5 cs to cs period 0 ~ 7
#define MSC_CS5_RRR           ( 3 <<(12+16))
// nCS5 faster device = 1
#define MSC_CS5_RBUFF         ( 0 <<(15+16))

#define MSC2_VAL ( MSC_CS4_RT | MSC_CS4_RBW \
                  | MSC_CS4_RDF | MSC_CS4_RDN | MSC_CS4_RRR | MSC_CS4_RBUFF \
                  | MSC_CS5_RT | MSC_CS5_RBW \
                  | MSC_CS5_RDF | MSC_CS5_RDN | MSC_CS5_RRR | MSC_CS5_RBUFF )

/*
*****
* PCMCIA and CF Interfaces 설정 : 사용하지 않음.
*****
*/
/* MECR 레지스터 */
// used count of PCMCIA Slot  0:1-slot  1:2-slot
#define MECR_NOS              ( 0 <<0 )
#define MECR_CIT              ( 0 <<1 ) // exist card  0:none  1:exist
#define MECR_VAL              (MECR_NOS | MECR_CIT)

```

```

/*MCMEM0 레지스터 */
#define MCMEM0_SET      ( 5 <<0 ) // 0 ~ 127   10~1280-nsec for 100MHz
#define MCMEM0_ASST     (15 <<7 ) // 0 ~ 31    10~320-nsec  for 100Mhz
#define MCMEM0_HOLD     ( 3 <<14) // 0 ~ 63    10~640-nsec  for 100Mhz

#define MCMEM0_VAL      (MCMEM0_SET | MCMEM0_ASST | MCMEM0_HOLD )

/*MCCATT0 레지스터 */
#define MCATT0_SET      ( 5 <<0 ) // 0 ~ 127   10~1280-nsec for 100MHz
#define MCATT0_ASST     (15 <<7 ) // 0 ~ 31    10~320-nsec  for 100Mhz
#define MCATT0_HOLD     ( 3 <<14) // 0 ~ 63    10~640-nsec  for 100Mhz
#define MCATT0_VAL      (MCATT0_SET | MCATT0_ASST | MCATT0_HOLD )

/*MCIO0 레지스터 */
#define MCIO0_SET       ( 5 <<0 ) // 0 ~ 127   10~1280-nsec for 100MHz
#define MCIO0_ASST      (15 <<7 ) // 0 ~ 31    10~320-nsec  for 100Mhz
#define MCIO0_HOLD      ( 3 <<14) // 0 ~ 63    10~640-nsec  for 100Mhz
#define MCIO0_VAL       (MCIO0_SET | MCIO0_ASST | MCIO0_HOLD )

/*MCMEM1 레지스터 */
#define MCMEM1_SET      ( 5 <<0 ) // 0 ~ 127   10~1280-nsec for 100MHz
#define MCMEM1_ASST     (15 <<7 ) // 0 ~ 31    10~320-nsec  for 100Mhz
#define MCMEM1_HOLD     ( 3 <<14) // 0 ~ 63    10~640-nsec  for 100Mhz
#define MCMEM1_VAL      (MCMEM1_SET | MCMEM1_ASST | MCMEM1_HOLD )

/*MCCATT1 레지스터 */
#define MCATT1ET        ( 5 <<0 ) // 0 ~ 127   10~1280-nsec for 100MHz
#define MCATT1_ASST     (15 <<7 ) // 0 ~ 31    10~320-nsec  for 100Mhz
#define MCATT1_HOLD     ( 3 <<14) // 0 ~ 63    10~640-nsec  for 100Mhz
#define MCATT1_VAL      (MCATT1ET | MCATT1_ASST | MCATT1_HOLD )

/*MCIO1 레지스터 */
#define MCIO1_SET       ( 5 <<0 ) // 0 ~ 127   10~1280-nsec for 100MHz
#define MCIO1_ASST      (15 <<7 ) // 0 ~ 31    10~320-nsec  for 100Mhz
#define MCIO1_HOLD      ( 3 <<14) // 0 ~ 63    10~640-nsec  for 100Mhz
#define MCIO1_VAL       (MCIO1_SET | MCIO1_ASST | MCIO1_HOLD )

/*
*****
*   Sync Static Memory 설정 : 사용하지 않음
*****
*/
#define SXMRS_VAL        0x00000000      /* NOT USED */
#define SXLCR_VAL        0x00000000      /* NOT USED */
#define FLYCNFG_VAL      0x01FE01FE      /* NOT USED */

/*
*****
*   PSSR(Power Manager Sleep Status 레지스터)
*****
*/
#define PSSR_RDH          (1<<5)          // GPIO input active = 1
#define PSSR_PH           (0<<4)          // Peripheral Control Hold = 1
#define PSSR_VAL          ( PSSR_RDH | PSSR_PH )

/*

```

```

*****
*   GPIO 설정
*****
*/
/*
* 부가기능 (Alternate Function) 을 사용하는 경우 부가기능 설정을 위하여
  입/출력설정도 해야 한다.
*/
#define GPIO_BIT(x)          ( 1 << ( x % 32 ) )
#define GPIO_ALT_FN_1_IN 0x1
#define GPIO_ALT_FN_1_OUT 0x1
#define GPIO_ALT_FN_2_IN 0x2
#define GPIO_ALT_FN_2_OUT 0x2
#define GPIO_ALT_FN_3_IN 0x3
#define GPIO_ALT_FN_3_OUT 0x3
#define GPIO_ALT_FN_(num,func) ( func << ( ( num % 16 ) * 2 ) )

/* nCS[1..5] 를 사용을 위한 설정 */
#define GP_nCS1      GPIO_BIT(15)    // chip select 1 : 출력 , 초기값 : HIGH
#define GP_nCS5      GPIO_BIT(33)    // chip select 5 : 출력 , 초기값 : HIGH
#define GP_nCS2      GPIO_BIT(78)    // chip select 2 : 출력 , 초기값 : HIGH
#define GP_nCS3      GPIO_BIT(79)    // chip select 3 : 출력 , 초기값 : HIGH
#define GP_nCS4      GPIO_BIT(80)    // chip select 4 : 출력 , 초기값 : HIGH
// alternate function nCS[1..5]
#define GP_nCS1_MD    GPIO_ALT_FN_(15, GPIO_ALT_FN_2_OUT)
#define GP_nCS5_MD    GPIO_ALT_FN_(33, GPIO_ALT_FN_2_OUT)
#define GP_nCS2_MD    GPIO_ALT_FN_(78, GPIO_ALT_FN_2_OUT)
#define GP_nCS3_MD    GPIO_ALT_FN_(79, GPIO_ALT_FN_2_OUT)
#define GP_nCS4_MD    GPIO_ALT_FN_(80, GPIO_ALT_FN_2_OUT)

/* बैंक5의 VLIO사용을 위한 설정 */
#define GP_RDY        GPIO_BIT(18)    // VLIO를 위한 Data Ready : 입력
#define GP_nPWE        GPIO_BIT(49)    // Write Signal : 출력 , 초기값 : HIGH
// alternate functions ready, pwe
#define GP_RDY_MD      GPIO_ALT_FN_(18, GPIO_ALT_FN_1_IN)
#define GP_nPWE_MD      GPIO_ALT_FN_(49, GPIO_ALT_FN_2_OUT)

/* Debug 용 LED를 위한 설정 */
#define GP_LED0        GPIO_BIT(2)    // led0 : 출력 , 초기값 : LOW
#define GP_LED1        GPIO_BIT(3)    // led0 : 출력 , 초기값 : LOW
#define GP_LED2        GPIO_BIT(4)    // led0 : 출력 , 초기값 : LOW
#define GP_LED3        GPIO_BIT(5)    // led0 : 출력 , 초기값 : LOW

/*
* 입/출력 방향 설정 : GPDR레지스터
*/
//GPIO 0~31
#define GPDR0_VAL ( GP_nCS1 | GP_LED0 | GP_LED1 | GP_LED2 | GP_LED3 )
//GPIO 32~63
#define GPDR1_VAL ( GP_nCS5 | GP_nPWE )
//GPIO 64~80
#define GPDR2_VAL ( GP_nCS2 | GP_nCS3 | GP_nCS4 )

/*
* 출력방향설정으로 했을때의 초기값 설정 : GPSR/GPCR레지스터

```



```

*/
#define GPSR0_VAL      GP_nCS1                //GPIO 0~31
#define GPSR1_VAL      GP_nCS5|GP_nPWE        //GPIO 32~63
#define GPSR2_VAL      GP_nCS2|GP_nCS3|GP_nCS4 //GPIO 64~80

#define GPCR0_VAL      GP_LED0|GP_LED1|GP_LED2|GP_LED3 //GPIO 0~31
#define GPCR1_VAL      0x00000000             //GPIO 32~63
#define GPCR2_VAL      0x00000000             //GPIO 64~80

/*
 * 부가기능 설정 : GPSR/GPCR레지스터
 */
#define GAFR0_L_VAL     GP_nCS1_MD            //GPIO 0~15
#define GAFR0_U_VAL     GP_RDY_MD             //GPIO 16~31
#define GAFR1_L_VAL     GP_nCS5_MD            //GPIO 32~47
#define GAFR1_U_VAL     GP_nPWE_MD            //GPIO 48~63
#define GAFR2_L_VAL     GP_nCS2_MD | GP_nCS3_MD //GPIO 64~79
#define GAFR2_U_VAL     GP_nCS4_MD            //GPIO 80
#endif

```

⑤ \$(BLOBTOP)/include/blob/arch/Makefile.am 파일에 우리의 헤더파일을 추가한다.

```
$(BLOBTOP)/include/blob/arch/Makefile.am
```

```

noinst_HEADERS = \
    assabet.h \
    kelb.h \
    ....

```

⑥ \$(BLOBTOP)/src/blob/Makefile.am 파일에 우리의 보드의 초기화 루틴 등 실행루틴이 들어있는 소스파일을 등록한다.

```
$(BLOBTOP)/include/blob/arch/Makefile.am
```

```

EXTRA_blob_rest_elf32_SOURCES = \
    assabet.c \
    kelb.c \
    ....

```

⑦ kelb.c 소스파일을 만들어 src/blob/와 src/diag/kelb.c에 추가한다.

아래의 소스는 Lubbock.c 파일을 수정한 것이다. CD롬의 blob에 있으니 복사해서 사용해도 된다.

```
kelb.c
```

```

#ifdef HAVE_CONFIG_H
# include <blob/config.h>
#endif

#include <blob/arch.h>

```

```
#include <blob/flash.h>
#include <blob/init.h>
#include <blob/serial.h>
#include <blob/command.h>
#include <blob/util.h>
```

플래쉬에 대한 정의를 한다.

여기서는 2x Intel 28F128J3A strataflash 의 32비트 데이터버스에 대한 설정이다.

```
#define KELB_FLASH_DRIVER (&intel32_flash_driver)
// #define KELB_FLASH_DRIVER (&intel16_flash_driver)
static flash_descriptor_t kelb_flash_descriptors[] =
{
    {
        size: 256 * 1024,
        num: 128,
        lockable: 1
    },
    {
        /* NULL block */
    },
};
```

해당 플래쉬 드라이버를 할당하는 함수를 만들고, 이를 초기화 리스트에 등록하여 하드웨어 초기화시 호출되도록 한다. intel32_flash_driver에 등록된 함수들은 NOR형 플래쉬 드라이버이며, NAND형 플래쉬를 쓰려는 경우 별도로 드라이버를 작성해야 한다.

```
static void init_kelb_flash_driver(void)
{
    flash_descriptors = kelb_flash_descriptors;
    flash_driver = KELB_FLASH_DRIVER;
}

__initlist(init_kelb_flash_driver, INIT_LEVEL_INITIAL_HARDWARE);
```

초기화시 자신의 시스템에 필요한 하드웨어 초기화를 수행하는 함수를 만들고 이를 초기화 리스트에 넣어 초기화시 호출되도록 한다. 여기서는 단지 시리얼 드라이버를 위한 pxa_serail_driver구조체를 할당하고 있으며, 이에 대해서는 소스를 따라가면서 후에 보도록 할 것이다.

```
static void kelb_init_hardware(void)
{
    serial_driver = &pxa_serial_driver;
}

__initlist(kelb_init_hardware, INIT_LEVEL_DRIVER_SELECTION);
```

아래는 플래쉬에 대한 명령에 대한 핸들러 함수를 정의하고, 이를 commandlist에 넣음으로써 쓰기, 지우기명령을 추가하고 있다.

```
/*
*****
* cmd_flash_write
*/
static int cmd_flash_write( int argc, char *argv[] )
{
    int ret = 0;
    u32 src = 0L;
    u32 dest = 0L;
    u32 len = 0L;
    if ( argc < 4 ) {
```

```

        ret = -1;
        goto DONE;
    }
    ret = strtou32( argv[1], &src );
    if ( ret < 0 ) {
        ret = -1;
        goto DONE;
    }
    ret = strtou32( argv[2], &dest );
    if ( ret < 0 ) {
        ret = -1;
        goto DONE;
    }
    ret = strtou32( argv[3], &len );
    if ( ret < 0 ) {
        ret = -1;
        goto DONE;
    }
    if ( len & (0x3) ) {
        len = (len>>2) + 1;
    } else {
        len = len>>2;
    }
    // K3 flash
    flash_unlock_region((u32*)dest, len);
    ret = flash_write_region( (u32 *)dest, (u32*)src, len );
    // K3 flash
    flash_lock_region((u32*)dest, len);

DONE:
    return ret;
}
static char flashwritehelp[] = "fwrite srcadr destadr size(bytes)\n"
"flash a memory region\n";
__commandlist( cmd_flash_write, "fwrite", flashwritehelp );

/*****
 * cmd_flash_erase
 */
static int cmd_flash_erase( int argc, char *argv[] )
{
    int ret = 0;
    u32    dest    = 0L;
    u32    len     = 0L;
    if ( argc < 3 ) {
        ret = -1;
        goto DONE;
    }
    ret = strtou32( argv[1], &dest );
    if ( ret < 0 ) {
        ret = -1;
        goto DONE;
    }
    ret = strtou32( argv[2], &len );
    if ( ret < 0 ) {
        ret = -1;
        goto DONE;
    }

```

```

    }
    if ( len & (0x3) ) {
        len = (len>>2) + 1;
    } else {
        len = len>>2;
    }
    // K3 flash
    flash_unlock_region((u32*)dest, len);

    ret = flash_erase_region( (u32 *)dest, len );

    // K3 flash
    flash_lock_region((u32*)dest, len);
DONE:
    return ret;
}

static char flasherasethelp[] = "ferase adr size(bytes)\n"
"erase a flash region\n";
__commandlist( cmd_flash_erase, "ferase", flasherasethelp );

```

⑧ \$(BLOBTOP)/utils/build/build_Makefile의 arch 변수에 우리보드의 이름을 추가한다.

```
$(BLOBTOP)/utils/build/build_Makefile
```

```

archs          = \
    assabet badge4 brutus creditlart h3600 idr\
    jornada720 lart neponset nesa pleb shannon\
    system3 kelb

```

⑨ \$(BLOBTOP)/include/blob/linux.h 파일에 아키텍처 번호를 세팅한다.

```
$(BLOBTOP)/include/blob/linux.h
```

```

#elif defined KELB
# define ARCH_NUMBER (194)

```

이것은 나중에 커널 부팅시 비교하게 되므로, 커널소스 수정시 맞춰줘야 한다.

다른 것들도 중요하지만 이것은 커널과 관련된 매우 중요한 부분이다.

⑩ 사용하는 시리얼포트 설정을 하기 위하여 \$(BLOBTOP)/src/lib/serial-pxa.c을 수정한다.

디폴트는 FFUART를 사용하고 있으며 다른 UART를 사용하고자 한다면 이 serial-pxa.c 파일을 수정해야 한다. 3. BLOB소스보기를 참조하라.

⑪ Ramdisk를 사용하고 있다면 \$(BLOBTOP)/src/blob/main.c파일의 아래부분의 주석을 없애야 한다.

```
$(BLOBTOP)/src/blob/main.c
```

```
146      /*          ←-없앰.  
147      if(blob_status.load_ramdisk)  
148          do_reload("ramdisk");  
149      */          ←-없앰.
```

3. BLOB 소스 보기

BLOB설치 과정 중 마지막에 잠깐 보았던 blob-start화일과 blob-rest화일을 상기하며 부트로더의 구조와 실행순서를 보도록 하자.

여기서 BLOB을 두개의 스테이지(스테이지1(start부분), 스테이지2(rest부분))로 구분하여 수행순서를 보도록 할 것이다.

그럼 스테이지 1과 스테이지 2로 나누어 하는 일을 간단히 보도록 하자.

스테이지 1에서 하는 일은 예외벡터의 등록, Supervisor 모드변환, 인터럽트 금지, CPU 코어 속도 설정, 메모리인터페이스 설정, 기타 하드웨어 초기화를 거쳐, 스테이지 2의 코드를 램영역으로 복사한 후 그곳으로 분기하여 스테이지 2를 실행한다. 이 스테이지1은 롬에서 실행되어 지고 두번째 스테이지를 램으로 복사하여 그곳으로 제어를 옮기는 데, 이유는 플래쉬에서 읽어 실행하는 것보다 램에 옮겨 실행하는 것이 메모리의 접근시간이 짧기 때문에 램으로 옮겨서 코드를 실행함으로써 프로그램 실행의 퍼포먼스를 높이기 위함이다.

이렇게 하여 스테이지 2로 넘어오면, C함수인 main()함수를 실행하기 위하여 스택 및 BSS 영역을 초기화 하고, main()함수를 실행한다. 이 main()함수에서는 시리얼 포트 초기화를 통하여 앞으로 디버깅용 메시지를 시리얼로 볼 수 있도록 해주며, 플래쉬롬에 있는 BLOB, Kernel, Ram Disk등 코드 및 데이터들을 해당 램영역으로 복사하게 된다. 그런 다음 시리얼로 부터의 키 입력을 기다리게 되며 일정시간동안 키 입력이 없다면, 자동부팅으로 넘어가 램에 복사된 커널코드(start_kernel)로 분기하여 실질적인 커널을 시작하게 된다. 키 입력이 있다면, 메뉴의 명령입력이 들어온 것으로 판단하고, 입력이 들어온 문자열과 같은 명령이 있는지 확인 후 있다면, 해당명령을 수행한다.

3.1. 링크스크립트

실질적인 코드를 보기 전에 우선 링커의 작업시에 규칙으로 사용되는 스크립트 화일들을 먼저 봄으로써 실행코드로 생성될 때 각각의 사용 영역들이 메모리의 어느 위치에 존재하게 되고, 어느 부분의 코드가 먼저 실행되는지 살펴볼 필요가 있다.

3.1.1. 스테이지 1을 위한 start-ld-script 파일

먼저 start-ld-script 파일을 보자. 이것은 ld(loader&linker)를 위한 스크립트화일로 Object 파일을 생성하는 규칙을 제공한다.

`$(BLOBTOP)/src/blob/start-ld-script 화일`

```

코드가 ELF32 의 Little Endian으로 생성함을 나타낸다.
13 OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
Architecture 가 ARM 임을 나타낸다.
14 OUTPUT_ARCH(arm)
엔트리를 _start 로 지정한다. 즉 이 _start 부터 실행되게 된다.
15 ENTRY(_start)
16 SECTIONS
17 {
    우선 0x00000000 번지부터 시작해서 4바이트 단위로 정렬된 text 섹션 이
    제일 먼저 나온다
18     . = 0x00000000;
19
20     . = ALIGN(4);
21     .text : { *(.text) }
22
23     . = ALIGN(4);
24     .rodata : { *(.rodata) }
25
26     . = ALIGN(4);
27     .data : { *(.data) }
28
29     . = ALIGN(4);
30     .got : { *(.got) }
31
32     . = ALIGN(4);
33     .bss : { *(.bss) }
34 }

```

3.1.2. 스테이지 2를 위한 rest-ld-script.in 파일

이 화일을 기초로 rest-ld-script화일이 만들어 지며, 두번째 스테이지에 해당하는 실행코드는 이 화일을 링커로 전달함으로써 만들어 지게 된다.

`$(BLOBTOP)/src/blob/rest-ld-script.in` 화일

```

인클루드할 화일을 선택한다. 여기서 arch.h파일소스 안에는 우리가 정의해 넣을 kelb.h
가 포함되어 있으며, 아래의 BLOB_ABS_BASE_ADDR은 그곳에서 정의된다.
24 #include <blob/config.h>
25 #include <blob/arch.h>
26
코드를 ELF32 의 Little Endian으로 생성한다
27 OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
아키텍는 ARM 이다.
28 OUTPUT_ARCH(arm)
엔트리로 _trampoline을 준다. 즉, 램으로 옮겨져 _trampoline부터 실행된다.
29 ENTRY(_trampoline)
30 SECTIONS
31 {

```

시작 위치를 정한다. 여기서는 `BLOB_ABS_BASE_ADDR`로 주었으며, 이 값은 `SDRAM`의 처음 위치인 `0xA0000000`으로 `include/blob/arch/kelb.h`에서 정의하고 있다. 따라서, 처음 스테이지에서 두번째 스테이지를 플래쉬로 부터 `SDRAM` 영역의 처음위치인 `0xA0000000`으로 복사한 후 그곳으로 점프하여 실행하게 되며, 이때 처음 실행되는 코드의 위치는 `trampoline`레이블이 된다.

```
32 . = BLOB_ABS_BASE_ADDR;
```

```
33
```

이하 각 영역에 대한 정의가 되어 있으며, 이들중 몇몇은 부팅 동작에서 사용되어 진다.

text 영역 설정

```
34 . = ALIGN(4);
35 .text : {
36     __text_start = .;
37     *(.text)
38     __text_end = .;
39 }
40
```

rodata 영역 설정

```
41 . = ALIGN(4);
42 .rodata : {
43     __rodata_start = .;
44     *(.rodata)
45     __rodata_end = .;
46 }
47
```

data 영역 설정

```
48 . = ALIGN(4);
49 .data : {
50     __data_start = .;
51     *(.data)
52     __data_end = .;
53 }
54
```

got 영역 설정

```
55 . = ALIGN(4);
56 .got : {
57     __got_start = .;
58     *(.got)
59     __got_end = .;
60 }
61
```

commandlist 영역 설정

이 영역은 메뉴들의 모음이 들어가게 되는 위치로 나중에 `BLOB` 메뉴항목에 명령이 추가되는 과정을 보면서 다루게 될 것이다.

```
62 . = ALIGN(4);
63 .commandlist : {
64     __commandlist_start = .;
65     *(.commandlist)
66     __commandlist_end = .;
67 }
68
```

initlist 영역 설정

초기화에 대한 영역으로 필요한 초기화가 있는 경우 매크로를 사용하여 이곳에 끼워넣게 되는데 이 경우 역시 `BLOB` 소스를 보면서 보게 될 것이다.


```

69     . = ALIGN(4);
70     .initlist : {
71         __initlist_start = .;
72         *(.initlist)
73         __initlist_end = .;
74     }
75
76     . = ALIGN(4);
77     .exitlist : {
78         __exitlist_start = .;
79         *(.exitlist)
80         __exitlist_end = .;
81     }
82
83     . = ALIGN(4);
84     .ptaglist : {
85         __ptagtable_begin = .;
86         *(.ptaglist)
87         __ptagtable_end = .;
88     }
89
90     . = ALIGN(4);
91     .bss : {
92         __bss_start = .;
93         /* first the real BSS data */
94         *(.bss)
95
96         /* and next the stack */
97         . = ALIGN(4);
98         __stack_start = .;
99         *(.stack)
100        __stack_end = .;
101        __bss_end = .;
102    }
103 }
104 }

```

exitlist 영역 설정
 빠져나올때 실행되어야 할 프로시저들이 등록된다.

ptaglist 영역 설정
 커널로 분기할 때 넘겨주어야 할 정보(TAG)들이 들어가는 곳으로 사용하지 않을 수 도 있다.

bss 영역 설정
 BSS영역은 C소스로 된 코드의 실행시 초기화되지 않은 변수들이 위치하는 영역으로 맨 마지막 섹션에 위치되어 있다.

3.2. 스테이지 1(Start.S)

첫번째 스테이지라 되어 있는 곳에서는 예외 벡터를 정의하고, 리셋의 원인을 찾아 해당 루틴을 실행하게 된다. 하드웨어 리셋의 경우 Normal Reset 루틴을 실행하게 되며, 여기서는 CPU 클럭, 시리얼 초기화등 하드웨어에 대한 초기화를 한다. 이후 두번째 스테이지 부분을 램 영역으로 재배치 한 후 그곳으로 점프하게 된다. 그럼 첫번째 스테이지를 소스를 따라가면서 보도록 하자.

3.2.1. 수행과정

start-ld-script에서 보았던 `_start` 엔트리포인트가 `start.S` 파일에 있으므로 이 파일부터 시작한다.

▣ 예외벡터들 정의와 `reset` 핸들러로 점프한다.



▣ CPU를 SVC32(Supervisor 모드)로 바꾸어 준다.



▣ `memsetup`의 호출로 코어클럭 및 메모리 설정을 한다.



▣ `ledinit` 루틴으로 분기하여 디버그용 LED 동작을 위한 초기화를 한다.



▣ `BLOB_START(BLOB_ABS_BASE_ADDR (0xA0000000))` 위치로 부터 처음 1MB에 대한 메모리에 대한 영역을 4K의 증가로 검사를 한다.



▣ 처음 4Kbyte 부분을 제외하고 `BLOB_START` 위치로 코드를 복사한 후, 즉 두번째 스테이지 부분을 램 영역으로 재배치한 후 재배치한 코드의 스타트업 코드인 `trampoline`으로 분기한다.

아래는 `start.S` 파일로 코어클럭, 메모리, GPIO 초기화 및 스테이지 2의 부분을 램 영역으로 재 배치한 후 그곳으로 분기하는 과정을 거친다.

`$(BLOBTOP)/src/blob/start.S` 소스중

Text 섹션임을 표시한다.

33 .text

.global로 선언된 `_start`를 보면 `start_ld_script`에서 엔트리 포인트를 `_start`로 해놓았으므로 여기서부터 시작할 것이라는 것을 알 수 있다.

48 .globl _start

▣ 예외 벡터들 정의와 `reset` 핸들러로의 점프

각각의 예외 벡터들을 정의해 놓았으며, `b` 명령으로 `reset` 레이블로 점프한다.

여기에서 주의할 점은 해당 예외 핸들러의 주소가 들어 있는 것이 아니라 `b` 명령으로 점프명령을 쓰고 있다.

Vector 주소에 대해서는 `PXA255`의 인터럽트부분을 참조하도록 한다.

36 .globl _start

37 _start: b reset

38 b undefined_instruction

@reset 핸들러로 점프하여 동작을 시작한다.

```

39         b        software_interrupt
40         b        prefetch_abort
41         b        data_abort
42         b        not_used
43         b        irq
44         b        fiq
46
47 BLOB_START: .word BLOB_ABS_BASE_ADDR
48
49     실제 동작을 시작하는 reset 레이블
50 reset:
51     ■ CPU를 SVC32(Supervisor 모드)로 바꾸어 준다.
52     CPSR레지스터의 하위 5비트에 접근하여 CPU를 SVC32(Supervisor 모드)로 바꾸어
53     준다.
54     - PXA255에 대해중 3항 XScale 코어레지스터와 예외편중 4.4. CPSR레지스터 참조.-
55     mrs     r0, cpsr
56     bic     r0, r0, #0x1f
57     orr     r0, r0, #0x13
58     msr     cpsr, r0
59
60     ■ memsetup의 호출로 코어클럭 및 메모리 설정을 한다.
61     memsetup으로 분기하여 메모리주파수 및 기타 클럭을 위한 CPU속도를 조정한다.
62     - [3.2.2. 호출된 중요 루틴들]중 A.memsetup 루틴 참조 -
63     bl      memsetup
64
65     ■ ledinit루틴으로 분기하여 디버그용 LED동작을 위한 초기화를한다.
66     ledinit으로 분기하여 해당 시스템에 디버그용 LED가 붙어 있다면 이를 초기화 한다.
67     - [3.2.2. 호출된 중요 루틴들]중 B.ledinit 루틴 참조 -
68     bl      ledinit
69
70     ■ BLOB_START(BLOB_ABS_BASE_ADDR (0xA0000000))위치로 부터 처음 1MB에
71     대한 메모리에 대한 영역을 4K의 증가로 검사를 한다.
72     r7에는 0x1000(4k)를 두고, r6에는 1M를 두어 4K 단위로 testram을 호출하여 해당 메
73     모리의 유/무를 확인하여 에러가 있을 경우 badram 루틴으로 분기하여 무한루프를
74     돌면서 led를 계속 깜빡이도록 한다.
75     여기서 BLOB_START는 BLOB_ABS_ADDR로 정의되어 있으며, 이는
76     include/blob/arch/kelb.h파일에서 정의된다. 이 값은 또한 $(BLOBTOP)/src/blob/ rest-
77     ld-script.in에서 두번째 스테이지의 처음 시작으로 설정되어 있음을 주목하자.
78     - [3.2.2. 호출된 중요 루틴들]중 C.testram 루틴 참조 -
79     mov     r7, #0x1000                @ r7=0x1000(4k)
80     mov     r6, r7, lsl #8              @r6= 4k << 2^8 = 1MB
81     ldr     r5, BLOB_START              @ r5=BLOB_START(0xA0000000)
82     @4KByte 씩 증가시켜며, 루프를 돌아 1M에 대한 메모리 검사를 한다.
83 mem_test_loop:
84     mov     r0, r5
85     bl      testram
86     teq     r0, #1
87     beq     badram
88
89     add     r5, r5, r7
90     subs    r6, r6, r7
91     bne     mem_test_loop

```

■ 처음 4Kbyte부분을 제외하고 BLOB_START위치로 코드를 복사한 후, 즉 두번째 스테이지 부분을 램 영역으로 재배치한 후 재배치한 코드의 스타트업 코드인 trampoline으로 분기한다.

플래쉬의 하위 4KByte를 제외하는 이유는 src/blob/Makefile.in중에 dd명령을 사용하여 아래와 같이 seek로 4Kbyte를 두어 rest부분의 코드는 4KByte의 Offset후에 놓이기 때문이다.

```
blob: blob-start blob-rest
    rm -f $@
    dd if=blob-start of=$@ bs=1k conv=sync
    dd if=blob-rest of=$@ bs=1k seek=4
    chmod +x $@
```

copy_loop루틴에서는 r0에는 복사할 소스의 시작주소, r1에는 복사할 타겟주소를, r2에는 복사할 데이터의 마지막 주소를 주고 ldmia와 stmia명령을 사용하여 루프를 돌면서 r0가 r2보다 작을 동안 복사를 한다. 이로써 모든 재배치가 끝났고, 마지막에서 LED를 OFF시키고, pc에 BLOB_START의 주소를 주어 이곳으로 분기한다. 분기하면 rest_ld_script에서 보듯이 _trampoline로 분기된다.

79 relocate:

```
80     adr     r0, _start                @blob의 처음 시작주소
81
82     /* 두번째 스테이지 로더를 재배치 한다. */
83     add     r2, r0, #(64 * 1024) /* blob의 최대크기를 64kB로 둔다. */
84     add     r0, r0, #0x1000          /* 처음 4KByte를 제외한다. */
85     ldr     r1, BLOB_START
86
87     /* r0 = 소스주소, r1 = 타겟주소, r2 = 소스 마지막 주소*/
91 copy_loop:
88     /*소스주소에서 데이터를 r3-r10레지스터로 읽어 들인다. r0=r0+8*/
92     ldmia   r0!, {r3-r10}
89     /*읽어들인 r3-r10레지스터의 내용을 타겟주소로 써 넣는다. r1=r1+8*/
93     stmia   r1!, {r3-r10}
94     cmp     r0, r2
95     ble     copy_loop /*r0가 r2보다 작을동안 반복하면서 복사*/.
96
97
98     /* 재배치한 두번째 스테이지 로더로 분기한다.*/
99     ldr     r0, BLOB_START
100    mov     pc, r0
```

3.2.2. 호출된 중요 루틴들

이 곳에서는 위의 과정을 거치며 호출하였던 함수들 중 몇가지를 살펴보도록 한다. 특히 memsetup 루틴은 시스템의 동작여부를 결정짓는 중요 루틴으로 이곳을 넘어가지 못하면 아무런 작업도 할 수 없으므로, 각자의 시스템에 맞도록 설정값들을 정확히 설정해야 한다.

A. memsetup 루틴

[bl memsetup] 명령으로 분기되는 이 루틴은 메모리를 setup 하는 루틴이다.

memsetup 루틴은 \$(BLOBTOP)/src/blob/memsetup-pxa.S에 존재하며, 이곳의 설정은 lubbock 보드의 설정에 관한 것이므로 보드에 맞게 수정을 가해야 한다.

하드웨어적인 설정에 대해서는 [PXA255에 대해중 8항 CLOCK관리자중 5. 주파

수변환과정]과 [PXA255에 대해중 10항 메모리컨트롤러중 7. 리셋후 메모리설정시퀀스]를 참조하도록 한다.

여기에 쓰인 상수정의들은 \$(BLOBTOP)/include/blob/arch/kelb.h에서 정의된다.

① 부트로더에서 GPIO 동작을 위한 초기 설정을 한다.



② 리셋후 메모리 설정시퀀스과정을 시작한다.
[PXA255에 대해중 10항 메모리컨트롤러중 7. 리셋후 메모리설정시퀀스 참조.]



③ Sleep 리셋인지 판단하여 Sleep 리셋이라면 그곳으로 점프하여 하던일을 다시 하도록 하며, 아니라면 주파수를 변경하기 위하여 인터럽트/클럭사용 금지한다.



④ 주파수 변환과정을 시작한다.
[PXA255에 대해중 8항 CLOCK관리자중 5. 주파수변환과정]을 참조하도록 한다



⑤ 주파수변환 시퀀스후 메모리동작을 재시작 시킨다.
주파수 변환 시퀀스 후 메모리는 반드시 재시작 되어야 한다.



⑥ 그 밖의 기타작업을 한 후 호출했던 곳으로 리턴한다.
메모리 내의 내용을 지우고, 인터럽트를 금지시키는 등의 일을 하고, 호출된 곳으로 리턴한다.

① 부트로더에서 GPIO 동작을 위한 초기 설정을 한다.

여기서는 GPIO에 대한 입/출력 방향 설정, 부가기능 설정, 에지디텍트 설정등 GPIO 전반에 관한 사항을 정의한다. 자신의 시스템에 맞게 설정하여야 하며, xxxx_VAL이라 표시된 값들은 특정레지스터에 쓰여질 값들로 kelb의 경우 \$(BLOBTOP)/include/blob/arch/kelb.h에 정의된 값들이다.

아래 GPIO 사용에 대해 나타내고 있으며, 부가기능을 사용하고자 할 때는 GPDR의 설정으로 GPIO의 입출력방향도 설정해야 함을 명심하자.

■ Debug 용 LED

사용 GPIO	사용용도	출력방향	초기값	부가기능사용
GP2,GP3 GP4,GP5	디버그용 LED	출력	OFF	사용하지 않음

■ Static Chip Select Signal

사용 GPIO	사용용도	출력방향	초기값	부가기능사용
GP15	Static Chip Select 1 (nCS1)	출력	ON	부가기능2번
GP33	Static Chip Select 5 (nCS5)	출력	ON	부가기능2번
GP78	Static Chip Select 2 (nCS2)	출력	ON	부가기능2번
GP79	Static Chip Select 3 (nCS3)	출력	ON	부가기능2번
GP80	Static Chip Select 4 (nCS4)	출력	ON	부가기능2번

■ Static 영역의 nCS5를 VLIO로 사용하기 위한 설정

사용 GPIO	사용용도	출력방향	초기값	부가기능사용
GP18	Data Ready 시그널	입력		부가기능1번
GP49	nPWE (VLIO는 이 시그널을 이용하여 쓰기를 수행한다.)	출력	ON	부가기능2번

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

```

18 #include <blob/arch.h>
19
20 .text
21
22 .globl memsetup @전역 변수로 globl로 선언하였음.
23 memsetup:      @memsetup 루틴시작
24
25     r10에 리턴할 주소를 넣어둔다.
26     mov     r10, lr
27 #if 0
28     중간 소스는 PXA250/210에 대한 사항으로 여기서는 관계가 없다.
29 #endif
30
31 #endif
32     다른 칩의 Chip Select를 활성화 하기 전에 GPIO설정을 해야 하므로 여기서는 이에
33     대한 설정을 한다.
34     우선 PSSR에 0x10을 씌으로써 PSSR의 PH비트를 클리어 한다.
35
36     ldr     r0,    =PSSR
37     ldr     r1,    =0x10
38     str     r1, [r0]
39
40     nop
41     nop
42     nop
43     nop
44     nop
45
46     nop
47     nop
48     nop
49     nop
50     nop
51
52     nop
53     nop
54     nop
55     nop
56     nop
57
58     nop
59     nop
60     nop
61     nop
62     nop
63
64     nop
65     nop
66     nop
67     nop
68     nop
69
70     nop
71     nop
72     nop
73     nop
74     nop
75

```

GPSR[0..2]에 대해 값을 줌으로써 특정포트에 대한 출력값을 설정 한다.. 1을 쓰면 출력으로 지정할 경우 HIGH시그널이 출력되고 0을 쓰면 아무런 영향이 없다. 자신의 시스템에 맞게 설정하도록 한다.(CS의 경우는 1을 써주어 선택되는 것을 막도록 한다)

[PXA255에 대해중 13항 GPIO편중 1.3. GPSR과 GPCR참조]

```

76      // GPSR - put a 1 on any of the GPIOs (0=unchanged, 1=drive 1)
77      ldr        r0,      =GPSR0
78      ldr        r1,      =GPSR0_VAL
79      str        r1, [r0]
80
81
82      ldr        r0,      =GPSR1
83      ldr        r1,      =GPSR1_VAL
84      str        r1, [r0]
85
86      ldr        r0,      =GPSR2
87      ldr        r1,      =GPSR2_VAL
88      str        r1, [r0]
89
90
```

GPCR[0..2]에 대해 값을 줌으로써 특정포트에 대한 출력값을 설정 한다.. 1을 쓰면 출력으로 지정할 경우 LOW시그널이 출력되고 0을 쓰면 아무런 영향이 없다. 자신의 시스템에 맞게 설정하도록 한다.

[PXA255에 대해중 13항 GPIO편중 1.3. GPSR과 GPCR참조]

```

91      //GPCR - put a 0 on any of the GPIOs (0=unchanged, 1=drive 0)
92      ldr        r0,      =GPCR0
93      ldr        r1,      =GPCR0_VAL
94      str        r1, [r0]
95
96      ldr        r0,      =GPCR1
97      ldr        r1,      =GPCR1_VAL
98      str        r1, [r0]
99
100     ldr        r0,      =GPCR2
101     ldr        r1,      =GPCR2_VAL
102     str        r1, [r0]
103
104
```

GPDR[0..2]에 대해 값을 줌으로써 포트들의 입출력 방향을 설정한다. 1을 쓰면 출력이며 0이면 입력이다. 자신의 시스템에 맞게 설정하도록 한다.

[PXA255에 대해중 13항 GPIO편중 1.1. GPDR참조]

```

105     //GPDR - put the GPIOs in the correct direction (0=in, 1=out)
106     ldr        r0,      =GPDR0
107     ldr        r1,      =GPDR0_VAL
108     str        r1, [r0]
109
110     ldr        r0,      =GPDR1
111     ldr        r1,      =GPDR1_VAL
112     str        r1, [r0]
113
114     ldr        r0,      =GPDR2
115     ldr        r1,      =GPDR2_VAL
116     str        r1, [r0]
117
```

GAFR[0..2]에 대해 설정하여 GPIO의 부가기능 사용여부를 설정한다. 자신의 시스템에 맞게 설정하도록 한다. nCS[1..4]등 시스템에 밀접한 부가기능도 있으므로 잘 살펴 설정하도록 한다.

[PXA255에 대해중 13항 GPIO편중 1.6. GAFR과 2.GPIO부가기능 참조]

```

118 //GAFR - setup the alternate functions (00=normal, 01=alt fuct 1, etc)
119 ldr      r0,      =GAFR0_L
120 ldr      r1,      =GAFR0_L_VAL
121 str      r1, [r0]
122
123 ldr      r0,      =GAFR0_U
124 ldr      r1,      =GAFR0_U_VAL
125 str      r1, [r0]
126
127 ldr      r0,      =GAFR1_L
128 ldr      r1,      =GAFR1_L_VAL
129 str      r1, [r0]
130
131 ldr      r0,      =GAFR1_U
132 ldr      r1,      =GAFR1_U_VAL
133 str      r1, [r0]
134
135 ldr      r0,      =GAFR2_L
136 ldr      r1,      =GAFR2_L_VAL
137 str      r1, [r0]
138
139 ldr      r0,      =GAFR2_U
140 ldr      r1,      =GAFR2_U_VAL
141 str      r1, [r0]
142
143 //PSSR = 0x20 clear the RDH bit in the PSSR
144 ldr      r0,      =PSSR
145 ldr      r1,      =0x20
146 str      r1, [r0]

```

② 리셋후 메모리 설정시퀀스과정을 시작한다.

리셋후 메모리 주파수 및 사용에 대한 설정을 한다. 이 과정은 [PXA255에 대해중 10항 메모리컨트롤러중 7. 리셋후 메모리설정시퀀스] 과정을 그대로 따라가므로 이 곳을 참조하면서 보도록 하자.

이 과정에서 GET_S23, GETS24등의 매크로를 사용하여 설정값을 읽어 오는데 이것은 lubbock보드의 하드웨어적 스위치 설정을 읽어 오는 것인데 우리의 KELB 보드에는 이 스위치가 없다. 따라서, 이들 매크로를 사용하지 않고 직접 값을 대입하도록 소스를 수정하겠다.

[1] 주파수변환 시작전 클럭 안정화를 위한 200uSec 정도의 시간을 대기한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

- SDCLK 클럭의 안정화를 위해 200uSec 정도 대기한다.

3.6864MHz의 Rising Edge에서 증가하는 OSCRE지스터를 클리어하고 0x300의 값과 비교함으로써 200usec정도의 루프를 돈다.

[PX255에 대해중 12항 Operating System Timer중 1.1. OSCRE지스터 참조.]

```

175      @ ---- Wait 200 usec
176      ldr r3, =OSCR                      @ OSCRE지스터의 주소를 읽어온다.
177      mov r2, #0                          @ OSCRE를 0으로 클리어한다.
178      str r2, [r3]
179      ldr r4, =0x300                      @ 실제로는 0x2E10이 200usec이므로 0x300 으로
주어 충분                                @ 하게 대기하도록 한다.
180 1:
181      ldr r2, [r3]
182      cmp r4, r2
183      bgt 1b                             @ OSCRE의 값이 0x300이 될때까지 대기
한다.

```

[2] Static Memory영역과 PCMCIA영역을 위한 설정을 한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

```

186      @ 메모리 컨트롤러의 Base 주소를 읽어온다.
187      ldr    r1, =MEMC_BASE
188
189 @*****
190 @ Step 1
191 @

```

[2].a. MSC[0..2]에 대한 설정을 한다.

MSC에 대한 내용은 [PX255에 대해중 10항 메모리컨트롤러중 5.1. MSC레지스터]를 참조하도록 하자. 여기서 Static Memory Bank 6개에 대한 설정을 한다. 설정되는 값은 각자의 시스템에 맞는 값이어야 하며, ezboard에서는 다음과 같이 값을 주고 있다.

비트	MSC0	설정내용
RT0[비트0:2]	0	CS0의 메모리타입 : NonBurst Rom 또는 플래쉬
RBW0[비트3]	0	CS0의 버스너비 : 32비트
RDF0[비트7:4]	13	CS0의 처음 접근시 유효데이터를 위한 지연 MEM 클럭 : 13 MEMCLK
RDN0[비트11:8]	2	CS0의 메모리의 처음 데이터기간 이후 연속되는 다음 유효 데이터 비트기간 : 2 MEMCLK
RRR0[비트 14:12]	2	CS0의 ROM/RAM 회복시간으로 CS와 다음 CS사이의 간격 : 2MEMCLK
RBUFF0[비트 15]	0	CS0의 빠른 디바이스인지 느린디바이스인지에 따른 설정으로 0은 느린디바이스 1은 빠른디바이스
RT1[비트16:18]	0	CS1의 메모리타입 : 0->NonBurst Rom 또는 플래쉬
RBW1[비트19]	1	CS1의 버스너비 : 1 -> 16비트
RDF1[비트 23:20]	13	CS1의 처음 접근시 유효데이터를 위한 지연 MEM 클럭 : 13 MEMCLK
RDN1[비트 27:24]	2	CS1의 메모리의 처음 데이터기간 이후 연속되는 다음 유효 데이터 비트기간 : 2 MEMCLK
RRR1[비트 30:28]	2	CS1의 ROM/RAM 회복시간으로 CS와 다음 CS사이의 간격 : 2MEMCLK
RBUFF1[비트 31]	0	CS1의 빠른 디바이스인지 느린디바이스인지에 따른 설정으로 0은 느린디바이스 1은 빠른디바이스
설정값		

```

192
193     @ 데이터 래치를 확실히 하기 위하여 쓰후에 다시 읽는다.
194     @
195     ldr    r2,    =MSC0_VAL
196     str    r2,    [r1, #MSC0_OFFSET]
197     ldr    r2,    [r1, #MSC0_OFFSET]
198

```

[2].a. MSC[0..2]에 대한 설정을 한다.

비트	MSC1	설정내용
RT0[비트0:2]	0	CS2의 메모리타입 : NonBurst Rom 또는 플래쉬
RBW0[비트3]	1	CS2의 버스너비 : 16비트
RDF0[비트7:4]	15	CS2의 처음 접근시 유효데이터를 위한 지연 MEM 클럭 : 15 MEMCLK
RDN0[비트11:8]	2	CS2의 메모리의 처음 데이터기간 이후 연속되는 다음 유효 데이터 비트기간 : 2 MEMCLK
RRR0[비트 14:12]	2	CS2의 ROM/RAM 회복시간으로 CS와 다음 CS사이의 간격 : 2MEMCLK
RBUFF0[비트 15]	0	CS2 빠른 디바이스인지 느린디바이스인지에 따른 설정으로 0은 느린디바이스 1은 빠른디바이스
RT1[비트16:18]	0	CS3의 메모리타입 : 0->NonBurst Rom 또는 플래쉬
RBW1[비트19]	1	CS3의 버스너비 : 1 -> 16비트
RDF1[비트 23:20]	15	CS3의 처음 접근시 유효데이터를 위한 지연 MEM 클럭 : 15 MEMCLK
RDN1[비트 27:24]	2	CS3의 메모리의 처음 데이터기간 이후 연속되는 다음 유효 데이터 비트기간 : 2 MEMCLK
RRR1[비트 30:28]	2	CS3의 ROM/RAM 회복시간으로 CS와 다음 CS사이의 간격 : 2MEMCLK
RBUFF1[비트 31]	0	CS3의 빠른 디바이스인지 느린디바이스인지에 따른 설정으로 0은 느린디바이스 1은 빠른디바이스
설정값		

```

199     @ write msc1
200     ldr    r2,    =MSC1_VAL
201     str    r2,    [r1, #MSC1_OFFSET]
202     ldr    r2,    [r1, #MSC1_OFFSET]
203

```

[2].a. MSC[0..2]에 대한 설정을 한다.

비트	MSC2	설정내용
RT0[비트0:2]	0	CS4의 메모리타입 : NonBurst Rom 또는 플래쉬
RBW0[비트3]	1	CS4의 버스너비 : 16비트
RDF0[비트7:4]	15	CS4의 처음 접근시 유효데이터를 위한 지연 MEM 클럭 : 15 MEMCLK
RDN0[비트11:8]	2	CS4의 메모리의 처음 데이터기간 이후 연속되는 다음 유효 데이터 비트기간 : 2 MEMCLK
RRR0[비트 14:12]	2	CS4의 ROM/RAM 회복시간으로 CS와 다음 CS사이의 간격 : 2MEMCLK
RBUFF0[비트 15]	0	CS4 빠른 디바이스인지 느린디바이스인지에 따른 설정으로 0은 느린디바이스 1은 빠른디바이스
RT1[비트16:18]	4	CS5의 메모리타입 : 4->VLIO
RBW1[비트19]	0	CS5의 버스너비 : 0 -> 32비트
RDF1[비트 23:20]	15	CS5의 처음 접근시 유효데이터를 위한 지연 MEM 클럭 : 15 MEMCLK
RDN1[비트 27:24]	2	CS5의 메모리의 처음 데이터기간 이후 연속되는 다음 유효 데이터 비트기간 : 2 MEMCLK
RRR1[비트 30:28]	2	CS5의 ROM/RAM 회복시간으로 CS와 다음 CS사이의 간격 : 2MEMCLK
RBUFF1[비트 31]	0	CS5의 빠른 디바이스인지 느린디바이스인지에 따른 설정으로 0은 느린디바이스 1은 빠른디바이스
설정값		

```

204         @ write msc2
205         ldr    r2, =MSC2_VAL
206         str    r2, [r1, #MSC2_OFFSET]
207         ldr    r2, [r1, #MSC2_OFFSET]
208

```

[2].b. PCMCIA에 대한 설정을 위하여 MECR, MCMEM0, MCMEM1, MCATT0, MCATT1, MCIO0, MCIO1을 설정한다.

```

209         @ write mecr
210         ldr    r2, =MECR_VAL
211         str    r2, [r1, #MECR_OFFSET]
212
213         @ write mcmem0
214         ldr    r2, =MCMEM0_VAL
215         str    r2, [r1, #MCMEM0_OFFSET]
216
217         @ write mcmem1
218         ldr    r2, =MCMEM1_VAL
219         str    r2, [r1, #MCMEM1_OFFSET]
220
221         @ write mcatt0
222         ldr    r2, =MCATT0_VAL
223         str    r2, [r1, #MCATT0_OFFSET]
224
225         @ write mcatt1
226         ldr    r2, =MCATT1_VAL
227         str    r2, [r1, #MCATT1_OFFSET]
228
229         @ write mcio0
230         ldr    r2, =MCIO0_VAL

```

```

231      str    r2, [r1, #MCIO0_OFFSET]
232
233      @ write mcio1
234      ldr    r2, =MCIO1_VAL
235      str    r2, [r1, #MCIO1_OFFSET]
236
237      @ fly-by-dma is defeatured on this part
238      @ write flycnfg
239      @ldr    r2, =FLYCNFG_SETTINGS
240      @str    r2, [r1, #FLYCNFG_OFFSET]
241
242

```

[2].c. MDREFR:K0RUN과 MDREFR:E0PIN를 설정하고 MDREFR:K0DB2를 설정한다. MDREFR:APD와 MDREFR:SLFRSH는 현재값을 유지한다. MDREFR:DRI를 설정한다. MDREFR:KxFREE를 해제한다.

MDREFR레지스터에 대한 내용은 [PXA255에 대해중 10항 메모리컨트롤러중 3.3.MDREFR레지스터]를 참조하도록 하자. 여기서는 MDREFR:DRI필드만 다시 써주고 있다.

비트	MDREFR	설정내용
DRI[비트11:0]	24	SDRAM의 리플래쉬 간격으로 MEM_CLK가 99.8MHz일때 24로 설정했다.
E0PIN[비트12]	0	SDCLK[0]의 클럭인에이블설정 : 디스에이블
K0RUN[비트13]	0	SDCLK[0] 핀 동작제어 : 디스에이블
K0DB2[비트14]	0	SDCLK[0]의 주파수설정 : 디스에이블
E1PIN[비트15]	1	SDCLK[1]의 클럭인에이블설정 : 인에이블
K1RUN[비트16]	1	SDCLK[1] 핀 동작제어 : 인에이블
K1DB2[비트17]	0	SDCLK[1]의 주파수설정 : MEM_CLK가 99.8Mhz로 설정했으므로 1:1로 동작시키기 위해 0으로 설정
K2RUN[비트18]	0	SDCLK[2] 핀 동작제어 : 디스에이블
K2DB2[비트19]	0	SDCLK[2]의 주파수설정 : 디스에이블
APD[비트20]	0	Auto-Power-Down 설정 : 디스에이블
예약[비트21]		
SLFRSH[비트22]	0	Self Refresh에 대한 설정으로 리셋시 반드시 클리어하여야 한다.
K0FREE[비트23]	0	SDRAM의 SDCLK0의 Free-Running동작 제어비트 : 0
K1FREE[비트24]	1	SDRAM의 SDCLK1의 Free-Running동작 제어비트 : 1
K2FREE[비트25]	0	SDRAM의 SDCLK2의 Free-Running동작 제어비트 : 0
예약[비트31:26]		
설정값		

```

243      @-----
244      @ 3rd bullet, Step 1
245      @
246
247      @ MDREFR 레지스터에 쓰여질 값을 읽어온다.
248      ldr    r3, =MDREFR_VAL
249
250      @ 읽어들인 값에서 DRI필드의 값만 빼낸다.
251      ldr    r2, =0xFFF

```

```

256      and    r3, r3, r2
257
258      @ MDREFR레지스터의 리셋후의 값을 읽어온다.
259      @
260      ldr     r4, [r1, #MDREFR_OFFSET]
261
262      @ 읽어들인 값에서 DRI 필드를 클리어 한다.
263      @
264      bic     r4, r4, r2
265
266      @ 위에서 MDREFR레지스터에 쓰여질 값중 뺀 DRI 필드의 값을 OR시켜
      @ 다시 써 넣는다. 즉, MDREFR레지스터에 리셋후 DRI 필드만 우선 다시
      @ 써 넣은 것이다.
267      @
268      orr     r4, r4, r3
272      str     r4, [r1, #MDREFR_OFFSET]
273
274      @ *r4에 mdrefr 값이 남아 있음을 기억하라.*

```

[3] Synchronous Static Memory를 포함하는 시스템에서, SXCNFG를 인에이블 비트를 포함하여 모든 쓰여지는 비트들을 설정하기 위하여 SXCNFG를 설정한다.

이 시스템에서 사용하지 않는 관계로 모두 주석처리 되어있으나, SMROM에 대한 설정을 하는 곳이다. 각자 보도록 하자.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

```

276 @*****
277 @ Step 2
278 @
279
280      @ fetch sxcnfg value
281      @
282      @ldr     r2, =0
283      @ write back sxcnfg
284      @str     r2, [r1, #SXCNFG_OFFSET]
285
286      @ if sxcnfg=0, don't program for synch-static memory
287      @cmp     r2, #0
288      @beq     1f
289
290      @program sxmrs
291      @ldr     r2, =SXMRS_SETTINGS
292      @str     r2, [r1, #SXMRS_OFFSET]
293
294

```

[4] SDRAM을 포함하는 시스템에서, 다음 단계에 의해서 SDRAM 컨트롤러를 전이한다.

MDREFR 레지스터의 값을 바꾸는데 이 것에 대해서는 [PXA255에 대해중 10항 메모리컨트롤러중 3.3. MDREFR레지스터]부분을 참조하도록 한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

r4에 이전의 mdrefr값을 가지고 있으며 우선 MDREFR레지스터의 Free-Running Clock 비트들(K0Free, K1Free, K2Free)를 클리어한다.

```
304     bic    r4, r4, #(0x00800000 | 0x01000000 | 0x02000000)
305
306     @ set K1RUN if bank 0 installed
307     @
```

[4].a. Bank0에 SDRAM이 설치되었다면, SDCLK1을 Enable하기 위하여 MDREFR레지스터의 K1RUN비트를 설정한다.

```
308     orr    r4, r4, #0x00010000
309
310 #ifdef LUBBOCK
```

이 부분은 LUBBOCK에 대한 설정을 하는 부분으로 MEMCLK와 SDCLK와의 비율을 정하는 곳으로 SDCLK가 100MHz가 넘지 않도록 설정한다.

이 부분의 코드가 필요에 의해 적용되어 저야 한다면 각자 보도록 하자.

```
412 #endif
    MDREFR레지스터에 값을 쓰고 다시 읽어온다.
```

```
414     @ write back mdrefr
415     @
416     str    r4, [r1, #MDREFR_OFFSET]
417     ldr    r4, [r1, #MDREFR_OFFSET]
418
```

[4].b. 현재 Self Refresh상태에서 [4].c.Power-Down 상태로 가기위하여 MDREFR:SLFRSH를 클리어하고 MDREFR레지스터에 이 값을 써 넣는다.

```
419     @ deassert SLFRSH
420     @
421     bic    r4, r4, #0x00400000
422
423     @ write back mdrefr
424     @
425     str    r4, [r1, #MDREFR_OFFSET]
426
```

[4].c. 현재 Power-Down상태에서 PWRDNX상태로 가기위하여 MDREFR:E1PIN을 설정하고 이 값을 MDREFR레지스터에 써 넣는다.

```
427     @ assert E1PIN
428     @
429     orr    r4, r4, #0x00008000
430
431     @ write back mdrefr
432     @
433     str    r4, [r1, #MDREFR_OFFSET]
434     ldr    r4, [r1, #MDREFR_OFFSET]
```

[4].d. 현재 PWRDNX상태에서 NOP로 가기 위하여 아무것도 하지 않는다.

```
435     nop
436     nop
```

[4].e. 위에서 SDRAM 파티션 페어에 대한 설정만 하였으며, 아직 인에이블 시키지 않도록 하기 위하여 MDCNFG레지스터의 DE[3,2,1,0]을 클러어 한다. SDRAM에 대해서는 해당 매뉴얼을 참조하도록 한다.

비트	MDCNFG	설정내용
DE0 [비트0]	1	SDRAM 뱅크 0인에이블
DE1 [비트1]	0	SDRAM 뱅크 1디스에이블
DWID0 [비트2]	0	SDRAM 0/1 뱅크에 대한 데이터버스 너비 : 32비트
DCAC0[1:0] [비트4:3]	01	SDRAM 1번 페이지에 대한 Column Address : 9 Column Address 비트
DRAC0[1:0] [비트6:5]	01	SDRAM 1번 페이지에 대한 Row Address : 12 Row Address
DNB0[비트7]	1	SDRAM 1번 페이지에 대한 내부뱅크수 : 4개의 SDRAM 내부뱅크
DTC0[1:0] [비트9:8]	11	SDRAM 1번 페어에 대한 접근 타이밍 $t_{RP}=3\text{clks}$, $CL=3$, $t_{RCD}=3\text{clks}$, $t_{RAS}(\text{min})=7\text{clks}$, $t_{RC}=1\text{clks}$
DADDR0 [비트10]	0	SDRAM 1번 페어를 일반 뱅크로 사용
DLATCH0 [비트11]	1	1로 설정.
DSA1111_0 [비트12]	1	SDRAM 1번 페어를 SA1111 Addressing이 아닌 일반 Addressing 모드로 사용
예약[15:13]		
DE0 [비트16]	0	SDRAM 뱅크 2 디스에이블
DE1 [비트17]	0	SDRAM 뱅크 3 디스에이블
DWID0 [비트18]	0	이 시스템에서는 SDRAM 2번 페어에 대해서는 사용하지 않음.
DCAC0[1:0] [비트20:19]	0	
DRAC0[1:0] [비트22:21]	0	
DNB0[비트23]	0	
DTC0[1:0] [비트25:24]	0	
DADDR0 [비트26]	0	
DLATCH0 [비트27]	1	
DSA1111_0 [비트28]	1	
예약[31:29]		

443 @ MDCNFG레지스터에 설정할 값을 읽어온다.

444 ldr r2, =MDCNFG_VAL

445

446 @ 모든 SDRAM 뱅크를 디스에이블 시킨다.

447 @

448 bic r2, r2, #(MDCNFG_DE0 | MDCNFG_DE1)

```

449      bic    r2, r2, #(MDCNFG_DE2 | MDCNFG_DE3)
450
451      @ बैंक 0/1에 대한 버스너비를 설정한다.
452      @
453      bic    r2, r2, #MDCNFG_DWID0      @0=32-bit
454
455
456      @ MDCNFG레지스터에 설정값을 써 넣는다.
457      @
458      str    r2, [r1, #MDCNFG_OFFSET]
459

```

[5] SDRAM을 포함하는 시스템에 대하여, SDRAM이 NOP 조건을 통하여 안정된 클럭을 받을 수있도록 하기 위하여 해당 SDRAM에 필요한 NOP Power-Up Waiting 기간을 기다린다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

```

    여기서는 200uSec동안 대기하며, 방식은 위에서 본 것과 같이 OSCR을 사용한다.
464      @ 200uSec 동안 대기한다.
465      @
466      ldr r3, =OSCR_BASE      @OSCR Base 주소를 읽어온다.
467      mov r2, #0              @OSCR레지스터에 0값을 써 넣는다.
468      str r2, [r3]
469      ldr r4, =0x300          @0x2E1이 200uSec이므로, 0x300을 주어 여유를 둔
    다.
470 1:
471      ldr r2, [r3]
472      cmp r4, r2
473      bgt 1b                  @OSCR의 값이 0x300이 될동안 대기한
    다.
474

```

[6] Coprocessor15의 Register1 Control레지스터의 Cache Enable비트를 클리어하여 Data Cache 비트(DCACHE)를 디스에이블 한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

```

    Coprocessor15의 Register1 Control Register의 모든비트를 클리어 시킴으로써, Cache
    를 끄고, MMU를 디스에이블하고, Alignment Fault를 디스에이블시키는 등의 일을 한
    다.
479
480      mov    r0, #0x78          @모든 비트를 클리어한다.
481      mcr    p15, 0, r0, c1, c0, 0      @이 값을 Coprocessor의 control
    Register에      @써 넣는다.

```

[7] SDRAM을 포함하는 시스템에서의 하드웨어 리셋에서, 디스에이블 된 SDRAM뱅크들중 어떤것으로든 리플래쉬 싸이클의 명시된 수(보통 8번)의 non-Burst 읽기

또는 쓰기접근을 하여 모든 4개의 SDRAM 뱅크들에 대하여 자동리플래쉬(CBR) 상태를 시작시킨다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

SDRAM에 쓰기를 8번하여 모든 뱅크들에 대하여 자동리플래쉬(CBR)을 시작시킨다.

```
489      ldr    r2, =SDRAM_BASE
490      str    r2, [r2]
491      str    r2, [r2]
492      str    r2, [r2]
493      str    r2, [r2]
494      str    r2, [r2]
495      str    r2, [r2]
496      str    r2, [r2]
497      str    r2, [r2]
```

[8] 퍼포먼스를 위하여 DCACHE 비트를 인에이블 시킨다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

여기서는 Enable 시키지 않고 있다. 필요하다면 여기서 인에이블 시키면 된다.

```
500 @ *****
501 @ Step 8: NOP (enable dcache if you wanna... we dont)
502 @
```

[9] SDRAM을 포함하는 시스템에서 MDCNFG:DE3:2, DE1:0을 설정함에 의하여 해당 SDRAM 파티션을 인에이블 한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

MDCNFG레지스터의 DEx비트들을 설정함으로써 SDRAM이 설치되어있는 Bank를 Enable 시킨다.

```
510      @메모리 컨트롤러의 Base 주소를 읽어온다.
511      @
512      ldr    r1, =MEMC_BASE
513
514      @MDCNFG레지스터의 값을 읽는다.
516      ldr    r3, [r1, #MDCNFG_OFFSET]
517
518      @여기서는 SDRAM Bank0,1에 대해서만 인에이블 시키고 있으며, 필요한 다
519      @큰 뱅크들도 여기서 Enable시키도록 한다.
520      orr    r3, r3, #MDCNFG_DE0
521      orr    r3, r3, #MDCNFG_DE1
522
523      @설정값을 MDCNFG 레지스터에 써 넣는다.
524      str    r3, [r1, #MDCNFG_OFFSET]
525
```

[10] SDRAM을 포함하는 시스템에서 MDMRS 레지스터에 설정값을 씌으로써 SDRAM의 모든 인에이블된 뱅크들에 MRS 명령을 시작한다. CAS 기간은 MDCNFG레지스터의 MTC0,2필드에서 설정된 값으로 설정되며, Burst 길이는 4로 설정된다. 이런 값들은 읽기 전용이므로 단지 0을 MDMRS 레지스터에 씌워서 동작한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

MDMRS_VAL은 0x0이며 이 값을 MDMRS 레지스터에 씌으로써 MRS 명령을 시작한다.

```
533      ldr    r2, =MDMRS_VAL
534      str    r2, [r1, #MDMRS_OFFSET]
535
```

[11] 옵션사항으로, SDRAM 또는 Synchronous Static Memory를 포함하는 시스템에서 MDREFR:APD를 설정함에 의해서 Auto-Power-Down을 인에이블 한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

옵션사항이며 Auto Power Down 모드를 활성화 시킨다. 여기서는 사용하지 않는다.

```
537 @*****
538 @ Step 11: Final Step
539 @
540 #ifdef 0
541
542     @!!! errata: don't enable auto power-down for A0
543     @get current value of mdrefr
544     @
545     @ldr    r3, [r1, #MDREFR_OFFSET]
546
547     @enable auto-power down
548     @
549     @orr    r3, r3, #MDREFR_APD
550
551     @write back mdrefr
552     @
553     @str    r3, [r1, #MDREFR_OFFSET]
554
555 #endif
```

③ Sleep 리셋인지 판단하여 Sleep 리셋이라면 그곳으로 점프하여 하던일을 다시 하도록 하며, 아니라면 주파수를 변경하기 위하여 인터럽트/클럭사용 금지한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

리셋 원인을 판단하여 Sleep Reset일 경우에 그곳으로 점프하여 하던일을 계속 수행하도록 한다.

```
561      ldr r0, =RCSR
562      ldr r1, [r0]
563      tst r1, #4
564
565      ldrne r0, =PSPR
566      ldrne r1, [r0]
567      movne pc, r1
```

인터럽트 컨트롤러에 있는 모든 인터럽트들을 디스에이블(Mask)한다.

```
573
574      @ ICLR(Interrupt Level Register)를 클리어한다.
575      @
576      mov     r1, #0
577      ldr     r2, =ICLR
578      str     r1, [r2]
579
580      @ 모든 인터럽트들을 마스크하여 디스에이블시킨다.
581      @
582      ldr     r2, =ICMR
583
584      str     r1, [r2]
585
586
```

다시 클럭을 설정하기 위하여 내부의 모든 부가 클럭들의 사용을 금지시킨다.

```
590      ldr     r1, =CKEN
591      mov     r2, #0
592      str     r2, [r1]
593
594
```

④ 주파수 변환과정을 시작한다.

[PXA255에 대해중 8항 CLOCK관리자중 5. 주파수변환과정]을 참조하도록 한다.

[1] CCCR레지스터의 값을 설정하여 L,M,N의 값을 설정한다.

여기서 SDCLK는 99.8 MHz를 사용하도록 하자.

\$ (BLOBTOP)/src/blob/memsetup-pxa.S 소스중

CCCR의 L,M,N의 값을 결정한다. 여기서 클럭조합이 가능한 모든 경우에 대한 체크를 하고 있으며, 이는 r0를 통하여 설정한다.

우리는 r0에 0x14를 두어 Memory Frequency는 99.5MHz, Run Mode Frequency는 398Mhz, 로 사용하도록 한다. SDCLK는 99.8 MHz가 된다.

만약, SDCLK가 99.8MHz보다 작다면 Memory Frequency는 SDCLK의 2배인 것으로 설정해야 한다.

CCCR_xxx상수에 대한 정의는 /include/blob/pxa.h 화일에 되어있다.

맨처음 언급했던 GET_S25, GET_S26등의 매크로 호출 부분으로 이 부분에서는 CCCR레지스터의 값을 선택하기 위한 값을 설정한다. 결과적으로 r0에 값이 설정되게 된다. 필요없는 부분이므로 주석처리 하도록 한다.

```
612      @ldr     r2, =FPGA_REGS_BASE
613      @bl      GET_S25
```

```

614      @bl      GET_
615
616      @orr      r0, r0, r3      @ concatenate S25 & S26 vals
617      @and      r0, r0, #0xFF
624      ldr      r2, =(CCCR_L27 | CCCR_M2 | CCCR_N10)      @ 디폴트:
{200/200/100}
여기서 저자는 CCCR_L27 | CCCR_M4 | CCCR_N10를 사용하기 위하여 r0에 0x14의
값을 강제로 주었다.
625      mov      r0, 0x14      @ 해당 클럭을 설정한다.
626      cmp      r0, #0x00      @ {100/100/100/100}
627      ldreq     r2, =(CCCR_L27 | CCCR_M1 | CCCR_N10)
628      cmp      r0, #0x01      @ {200/100/100/100}
629      ldreq     r2, =(CCCR_L27 | CCCR_M1 | CCCR_N20)
630      cmp      r0, #0x02      @ {300/100/100/100}
631      ldreq     r2, =(CCCR_L27 | CCCR_M1 | CCCR_N30)
632
633      cmp      r0, #0x03      @ {118/118/118/59} *
634      ldreq     r2, =(CCCR_L32 | CCCR_M1 | CCCR_N10)
635      cmp      r0, #0x04      @ {236/118/118/59} *
636      ldreq     r2, =(CCCR_L32 | CCCR_M1 | CCCR_N20)
637      cmp      r0, #0x05      @ {354/118/118/59} *
638      ldreq     r2, =(CCCR_L32 | CCCR_M1 | CCCR_N30)
639
640      cmp      r0, #0x06      @ {133/133/133/66} *
641      ldreq     r2, =(CCCR_L36 | CCCR_M1 | CCCR_N10)
642      cmp      r0, #0x07      @ {266/133/133/66} *
643      ldreq     r2, =(CCCR_L36 | CCCR_M1 | CCCR_N20)
644      cmp      r0, #0x08      @ {399/133/133/66} *
645      ldreq     r2, =(CCCR_L36 | CCCR_M1 | CCCR_N30)
646
647      cmp      r0, #0x09      @ {148/148/148/74} *
648      ldreq     r2, =(CCCR_L40 | CCCR_M1 | CCCR_N10)
649      cmp      r0, #0x0A      @ {296/148/148/74} *
650      ldreq     r2, =(CCCR_L40 | CCCR_M1 | CCCR_N20)
651
652      cmp      r0, #0x0B      @ {166/166/166/83} *
653      ldreq     r2, =(CCCR_L45 | CCCR_M1 | CCCR_N10)
654      cmp      r0, #0x0C      @ {332/166/166/83} *
655      ldreq     r2, =(CCCR_L45 | CCCR_M1 | CCCR_N20)
656
657      cmp      r0, #0x0D      @ {200/200/100/100}
658      ldreq     r2, =(CCCR_L27 | CCCR_M2 | CCCR_N10)
659      cmp      r0, #0x0E      @ {300/200/100/100}
660      ldreq     r2, =(CCCR_L27 | CCCR_M2 | CCCR_N15)
661      cmp      r0, #0x0F      @ {400/200/100/100}
662      ldreq     r2, =(CCCR_L27 | CCCR_M2 | CCCR_N20)
663
664      cmp      r0, #0x10      @ {236/236/118/59} *
665      ldreq     r2, =(CCCR_L32 | CCCR_M2 | CCCR_N10)
666
667      cmp      r0, #0x11      @ {266/266/133/66} *
668      ldreq     r2, =(CCCR_L36 | CCCR_M2 | CCCR_N10)
669
670      cmp      r0, #0x12      @ {295/295/148/74} *
671      ldreq     r2, =(CCCR_L40 | CCCR_M2 | CCCR_N10)
672
673      cmp      r0, #0x13      @ {332/332/166/83} *

```

```

674      ldreq    r2, =(CCCR_L45 | CCCR_M2 | CCCR_N10)
675
676      cmp      r0, #0x14                      @ {398/398/100/100} C0/Dalhart
permissible
677      ldreq    r2, =(CCCR_L27 | CCCR_M4 | CCCR_N10)

```

[2] CCCR레지스터의 값을 써넣고, 32.768KHz 오실레이터를 인에이블 시킨후 주파수 변환 시퀀스를 시작한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

```

.위에서 설정한 값을 CCCR 레지스터에 써 넣는다.
681      @... CCCR 레지스터에 위의 값을 써 넣는다.
682      @
683      ldr      r1, =CCCR
684      str      r2, [r1]
685
OSCC레지스터의 OON 비트에 1값을 씌으로써 32.768KHz 오실레이터를 인에이블
시킨다.
688      ldr      r1, =OSCC
689      mov      r2, #OSCC_OON
690      str      r2, [r1]
691
692
693      @ NOTE: spin here until OSCC.OOK get set,
694      @       meaning the PLL has settled.
695      @
32.768KHz 오실레이터가 안정화되기를 기다리기 위하여 OSCC의 OOK비트를 체크
한다
697      ldr      r2, [r1]
698      ands      r2, r2, #1
699      beq      60b

Coprocessor 14의 CCLKCFG레지스터의 TURBO비트를 사용하여 Turbo모드의 사용
여부를 결정하고, FCS비트를 세팅함으로써 주파수 변환과정을 들어간다.
Coprocessor 14의 CCLKCFG레지스터에 관해서는 [PXA255에 대해중 8항 CLOCK관
리자중 4. Coprocessor14의 CCLKCFG레지스터]를 참조하라.
이곳에서도 GET_SXX 매크로가 사용되는데 필요 없는 부분이므로 주석처리하도록
한다.
713      mov r1, #2                      @ 주파수변환을 시작시키는 FCS(Frequency Change
Bit)을 @설정한다.
714
715      @ldr      r2, =FPGA_REGS_BASE
716      @bl      GET_S24                  @ r0, r2
터보모드를 사용하지 않도록 하기 위하여 이 곳에서 강제로 r0에 0x01값을 주었다.
0x00을 준다면 터보모드로 들어가게 된다.
717      mov r0, #0x01
718      cmp r0, #0x0
719      moveq r1, #3                      @ r0가 0이라면 TURBO비트와 FCS비트를 동시에
셋팅한다.
720
721
722      mcr p14, 0, r1, c6, c0, 0        @ CP14의 CCLKCFG레지스터에 값을 써 넣는

```

다.
723

⑤ 주파수변환 시퀀스후 메모리동작을 재시작시킨다.

주파수 변환 작업후 메모리 컨트롤러는 반드시 다시 시작 시켜야 한다. 이곳에서는 이 작업을 하고 있다. 이는 위의 메모리 설정 시퀀스를 참조하도록 하자.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

```
.메모리 컨트롤러의 Base Address를 얻어온다.
736     ldr    r1, =MEMC_BASE
737
MDREFR레지스터를 설정한다.
738     @ MDREFR레지스터의 값을 읽어온다.
740     ldr    r2, [r1, #MDREFR_OFFSET]
741
742     @ E0PIN, E1PIN비트를 클리어한다.
744     bic    r3, r2, #(MDREFR_E0PIN | MDREFR_E1PIN)
745
746     @ E0PIN, E1PIN비트가 클러어된 값을 MDREFR에 써 넣는다.
748     str    r3, [r1, #MDREFR_OFFSET]
749
750     @ E0PIN, E1PIN비트를 셋하여 다시 써 넣는다.
752     str    r2, [r1, #MDREFR_OFFSET]
753
MDREFR레지스터를 설정한다.
754     @ MDCNFG레지스터의 값을 읽어온다.
756     ldr    r3, [r1, #MDCNFG_OFFSET]
757
758     @ 모든 SDRAM banks들을 금지시킨다.
760     bic    r3, r3, #(MDCNFG_DE0 | MDCNFG_DE1)
761     bic    r3, r3, #(MDCNFG_DE2 | MDCNFG_DE3)
762
763     @ 이 값을 MDCNFG레지스터에 쓴다.
765     str    r3, [r1, #MDCNFG_OFFSET]
766
767     @ 모든 बैं크에 대하여 CBR 리플래쉬 싸이클을 발생시키기 위하여 메모리에
    @ 접근한다. (8번)
769     ldr    r2, =SDRAM_BASE
770     str    r2, [r2]
771     str    r2, [r2]
772     str    r2, [r2]
773     str    r2, [r2]
774     str    r2, [r2]
775     str    r2, [r2]
776     str    r2, [r2]
777     str    r2, [r2]

754     @ MDCNFG레지스터의 값을 읽어온다.
781     ldr    r3, [r1, #MDCNFG_OFFSET]
782
783     @ SDRAM बैं크 0을 인에이블 시킨다. (설치된 बैं크들을 모두)
785     orr    r3, r3, #MDCNFG_DE0
786
```

```

787      @ MDCNFG 레지스터에 이 값을 써서 해당 뱅크를 인에이블 시킨다.
788      @
789      str    r3, [r1, #MDCNFG_OFFSET]
790
791      MRS 명령을 주기위하여 MDMRS 레지스터에 쓴다.
793      ldr    r2, =MDMRS_VAL
794      str    r2, [r1, #MDMRS_OFFSET]
795
796      아래는 Auto Power Down 모드를 사용할 경우 설정하는 것으로 여기서는 모두 주석처리 되어 있다.
798      @ errata: don't enable auto power-down
799      @ get current value of mdrefr
800      @ ldr    r3, [r1, #MDREFR_OFFSET]
801      @ enable auto-power down
802      @ orr    r3, r3, #MDREFR_APD
803      @ write back mdrefr
804      @ str    r3, [r1, #MDREFR_OFFSET]

```

⑥ 그 밖의 기타작업을 한 후 호출했던 곳으로 리턴한다.

메모리 내의 내용을 지우고, 인터럽트를 금지시키는 등의 일을 하고, 호출된 곳으로 리턴한다.

\$(BLOBTOP)/src/blob/memsetup-pxa.S 소스중

```

SDRAM 메모리의 내용을 모두 지우도록 한다.
816      ldr    r11, =0xa0000000 //SDRAM의 Base Address를 적는다.
817      ldr    r12, =0x04000000 // 지울 메모리의 크기를 적는다.
818      mov    r8, r12           // DRAM의 크기를 r8에 저장한다.
819      mov    r0, #0           // r0-7의 레지스터에 모두 0으로 쓴다.
820      mov    r1, #0
821      mov    r2, #0
822      mov    r3, #0
823      mov    r4, #0
824      mov    r5, #0
825      mov    r6, #0
826      mov    r7, #0
827 10:      @ 이 루프에서 32바이트씩 위에서 설정한 SDRAM 영역에 0을 씌으로써 초기화한다.
828      subs  r12, r12, #32     // 32 bytes/line
829      stmia r11!, {r0-r7}
830      beq   15f
831      b     10b
832 15:
833
834      모든 인터럽트를 금지한다.
835      ldr    r0, =ICMR
836      mov    r1, #0
837      str    r1, [r0]
838
839      소프트웨어와 데이터 Break 포인트를 금지한다.
840      mov    r0, #0
841      mcr    p15, 0, r0, c14, c8, 0 // ibcr0
842      mcr    p15, 0, r0, c14, c9, 0 // ibcr1

```

```

843      mcr      p15,0,r0,c14,c4,0 // dbcon
844

```

모든 디버깅 기능들을 인에이블 시킨다.

```

846      mov      r0,#0x80000000
847      mcr      p14,0,r0,c10,c0,0 // dcsr
848
849 ret:
850

```

맨처음 r10에 리턴할 주소를 놓았었으며, 여기서 그곳으로 리턴한다. start.S에서 호출 되었으므로 그곳으로 분기한다.

```

851      mov      pc, r10

```

B. ledinit루틴

b는 jmp 명령, bl은 call명령으로 이해하면 되겠다. ledinit 레이블은 \$(BLOBTOP)/src/blob/ledasm.S 파일에 .global ledinit으로 export 되어있고 소스는 아래와 같다. 이 화일에는 이밖에 led_on루틴, led_off루틴, led_blink루틴 등 LED관련 루틴들이 들어 있다. 이 디버깅용 LED는 말 그대로 디버깅을 위해 CPU의 동작상태를 보기 위하여 설치된 LED로 웬만한 보드에는 다 달려 있을 것이다.

이 LED는 부트로더 디버깅시 더 없이 좋은 디버깅용 툴로 사용될 수 있으며, 자신의 시스템에 맞도록 코드들을 수정해야 한다. 만약, GPIO의 입출력방향 및 부가설정기능을 memsetup-pxa.S에서 이미 고려하여 했다면 관계가 없겠지만, 설정하지 않았다면, 여기서는 디버깅용 LED를 위하여 GPIO에 대한 설정을 해야 한다.. 또한, led_on, led_off루틴도 또한 자신의 시스템에 맞게 변경해야 한다.

\$(BLOBTOP)/src/blob/ledasm.S 소스중

__ASSEMBLY__ 정의는 include/blob/arch/kelb.h파일에서 보면 사용되는 매크로가 다르다는 것을 알 수 있다. kelb.h를 참조하라. 여기서는 어셈블언어 사용이므로 이 정의를 해주도록 한다. C 코드에서는 이 정의를 빼야 한다.

```

31 #define __ASSEMBLY__
32
33 #ifdef HAVE_CONFIG_H
34 # include <blob/config.h>
35 #endif
36
37 #include <blob/arch.h>
38
39
40
41 .text

```

LED 초기화 : LED를 위한 GPIO 입출력방향, 부가기능을 초기화 하지 않았다면, 이곳에서 하도록 한다.

```

44 .globl ledinit
45 ledinit:

```

@출력 방향을 위한 GPDR설정, 부가기능 제거를 위한 GAFR설정, 초기 LED상태 설정을 위한 GPSR, GPCR설정을 하도록 한다.


```

75      mov     pc, lr
      Led On 루틴
77 .globl led_on
79 led_on:
      @LED를 켜는 함수로 시스템에 따라 GPSR 또는 GPCR을 설정하여 LED를 ON
      시키도록 한다.
80      mov     pc, lr

      Led Off 루틴
85 .globl led_off
87 led_off:
      @LED를 끄는 함수로 시스템에 따라 GPSR 또는 GPCR을 설정하여 LED를 OFF시키도록
      한다.
88      mov     pc, lr

      Blink 루틴 : Led 깜박이 기능을 위한 루틴으로 위의 led_on, led_off함수를 사용하므
      로 수정할 곳은 없으며, r0에 몇번 깜박일 숫자를 넣어줌으로써 그 숫자만큼 깜박이
      게 한다.
92 wait_loop:
93      mov     r0, #0x400000
94 wait_loop1:
95      subs    r0, r0, #1
96      bne     wait_loop1
97      mov     pc, lr
98
99
100 blinky:
101      /* blink LED. clobbers r0 and r1 */
102      mov     r2, lr
103      bl      led_on
104      bl      wait_loop
105      bl      led_off
106      bl      wait_loop
107      mov     pc, r2
108
109
110 .globl led_blink
111      /* blinks LED r0 times */
112      /* clobbers r0 - r4 */
113 led_blink:
114      mov     r4, lr
115      mov     r3, r0
116
117      /* switch LED off if it wasn't already off */
118      bl      led_off
119      bl      wait_loop
120
121 blink0:
122      bl      blinky
123      subs    r3, r3, #1
124      bne     blink0
125      mov     pc, r4

```

C. testram 루틴

testmem.S와 testmem2.S두개의 파일에 메모리를 검사하는 루틴인 testram루틴

이 있는데 하는 일은 같고 다만 레지스터를 저장했다가 나올때 복구를 하는지 안하는지의 차이점을 갖는다. C함수 루틴에서는 레지스터의 값을 저장/복구의 과정을 거치는 testmem2.S루틴을 불러야 할 것이다.

아래의 소스는 testmem.S파일중에 testram루틴에 대한 소스이다.

\$(BLOBTOP)/src/blob/testmem.S 소스중

```

34 .globl testram
35     @ r0 = 테스트할 시작 주소
36     @ 리턴값 r0 = 0 - 램존재, r0 = 1 - 램에러
37     @ clobbers r1 - r4
38 testram:
39     ldmia    r0, {r1, r2}    @ r0가 가리키는 곳에 원래 저장된 2개의 32비트 값을 r1과 r2로 읽어들인다.

40     mov     r3, #0x55        @ 처음 워드가 될 r3에 0x55를 둔다
41     mov     r4, #0xaa        @ 두번째 워드가 될 r4에는 0xaa를 둔다
42     stmia   r0, {r3, r4}     @ r0가 가리키는 곳에 r3와 r4의 값을 쓴다
43     ldmia   r0, {r3, r4}     @ 쓰여진 값을 다시 읽어 잘 쓰여졌는지 확인한다
44     teq     r3, #0x55
45     teqeq   r4, #0xaa
46     bne     bad              @ 만약 다르다면 bad :로 제어를 옮긴다.
47     mov     r3, #0xaa        @ 아래부분은 만약 0x55와 0xaa가 원래 그곳에 있던 값과 @같을수 있으므로 이들의 값을 뒤바꾸어 다시 확인한다
48     mov     r4, #0x55        @ 0x55 to second
49     stmia   r0, {r3, r4}
50     ldmia   r0, {r3, r4}     @ read it back
51     teq     r3, #0xaa        @ do the values match
52     teqeq   r4, #0x55
53 bad:     stmia   r0, {r1, r2} @ 원래 있던 값을 복구한다
54     moveq   r0, #0           @ 값이 제대로 쓰여졌었다면, r0에 0을 두어 에러가 없었음을 @나타낸다
55     movne   r0, #1           @ 램이 없다면, r0에 1을두어 에러가 있음을 나타낸다.
56     mov     pc, lr          @리턴한다.

```

3.3. 스테이지 2의 Trampoline.S

위에서 두번째 스테이지를 재 배치한 후 분기하는 최초위치는 `_trampoline`이 되며 이 루틴은 `.trampoline.S` 파일에 존재한다. 이 루틴에서는 C 코드인 `main()`으로 분기 하기 위하여, BSS영역과 스택포인터를 초기화 시킨 후 C 코드인 `main()`으로 점프하는 일을 한다.

`$(BLOBTOP)/src/blob/trampoline.S` 소스중

```

24 #ifdef HAVE_CONFIG_H
25 # include <blob/config.h>
26 #endif
27
28 .text
29
30 .globl _trampoline
31 _trampoline:
32     ■ BSS 섹션을 초기화한다.
33         ldr    r1, bss_start
34         ldr    r0, bss_end
35         sub    r0, r0, r1
36
37         /* r1 = 시작주소 */
38         /* r0 = 바이트수 */
39         mov    r2, #0
40
41 clear_bss:
42         stmia   r1!, {r2}
43         subs    r0, r0, #4
44         bne     clear_bss
45
46     ■ 스택포인터를 초기화한다.
47         ldr    r0, stack_end
48         sub    sp, r0, #4
49
50     ■ C코드인 main함수로 분기한다.
51         bl     main
52     넘어오면 안되지만 만약 이곳으로 넘어온다면 단지 다시한번 _trampoline루틴을 호출
53     한다
54         b      _trampoline
55
56 bss_start:    .word    __bss_start
57 bss_end:      .word    __bss_end
58 stack_end:   .word    __stack_end

```

3.4. 스테이지 2의 main함수

계속해서 보게 되겠지만, 이 `main()`함수에서는 각종 초기화, 시리얼포트 초기화, 램

디스크, 커널을 램영역으로 이전, 자동부팅 또는 명령을 받아 해석/처리하는 단계를 밟는다.

아래에서 이들의 작업과정에 대하여 천천히 살펴보도록 하자.

3.4.1. main() 함수

■ `init_subsystems()`호출로 `init`영역의 초기화 함수들을 실행시킨다.



■ 시리얼 포트설정, 파라미터, 램디스크를 로드할 지등의 설정을 위하여 `blob_status`구조체의 각 필드를 설정한다.



■ `serial_init()`함수의 호출로 위에서 설정한 환경에 맞도록 시리얼 포트를 초기화 한다.



■ GPL 문자들을 시리얼로 출력한다.



■ 메모리의 양을 얻는다.



■ `do_reload()`함수 호출로 “blob”, “kernel”, “ramdisk” 의 인자를 주어 플래쉬로 부터 램으로 커널과 램디스크를 로드한다.



■ 지정한 시간(여기서는 10초)동안의 키입력을 기다린다.



■ 키가 안눌렸다면 자동부팅을 시작한다. 이때는 커널시작시 사용하는 각 태그정보들을 설정하고, 커널의 스타트업루틴으로 분기한다.

■ 기간내에 키가 눌렸다면 키를 입력받아 메뉴항목과 비교하여 해당 루틴을 실행한다.

아래 `main()`함수를 소스를 보이며, 수행순서는 위의 수행단계표와 같다. 주석을 살펴피며 천천히 살펴보도록 하자.

`$(BLOBTOP)/src/blob/main.c` 의 `main()`함수 소스중

```
73 int main(void)
74 {
75     int numRead = 0;
```

```

76     int done=0;
77     char commandline[MAX_COMMANDLINE_LENGTH];
78     int i,count;
79     int retval = 0;
80 #ifdef PARAM_START
81     u32 conf;
82 #endif
83
84     ■ init_subsystems()호출로 init영역의 초기화 함수들을 실행시킨다.
85     - [4.3.2. 호출된 중요 루틴들]중 A.init_subsystems 함수 참조 -
86     init_subsystems();
87
88     ■ 시리얼 포트설정, 파라미터, 램디스크를 로드할 지등의 설정을 위하여
89     blob_status구조체의 각 필드를 설정한다.
90     - [4.3.2. 호출된 중요 루틴들]중 B.Blob 구조체(blob_status_t) 참조 -
91     blob_status.paramType = fromFlash;
92     blob_status.kernelType = fromFlash;
93     blob_status.ramdiskType = fromFlash;
94     blob_status.downloadSpeed = baud_115200;
95     blob_status.terminalSpeed = baud_115200;
96     blob_status.load_ramdisk = 1; @램디스크로드
97     blob_status.cmdline[0] = '\0';
98     blob_status.boot_delay = 10;
99
100    /* call serial_init() because the default 9k6 speed might not
101       be what the user requested */
102    #if defined(H3600) || defined(SHANNON) || defined(IDR) || defined(BADGE4) ||
103    defined(JORNADA720)
104        blob_status.terminalSpeed = baud_115200; /* DEBUG */
105    #endif
106    #if defined(PT_SYSTEM3)
107        blob_status.terminalSpeed = baud_38400;
108    #endif
109
110    ■ serial_init()함수의 호출로 위에서 설정한 환경에 맞도록 시리얼 포트를
111    초기화 한다.
112    - [4.3.2. 호출된 중요 루틴들]중 B.serial_init() 함수 참조 -
113    serial_init(blob_status.terminalSpeed);
114
115    /* parse the core tag, for critical things like terminal speed */
116    #ifdef PARAM_START
117        //parse_ptag((void *) PARAM_START, &conf);
118    #endif
119
120    ■ GPL 문자들을 시리얼로 출력한다.
121    /* Print the required GPL string */
122    SerialOutputString("\nConsider yourself LARTed!\n\n");
123    SerialOutputString(version_str);
124    SerialOutputString("Copyright (C) 1999 2000 2001 "
125        "Jan-Derk Bakker and Erik Mouw\n");
126    SerialOutputString(PACKAGE " comes with ABSOLUTELY NO WARRANTY; "
127        "read the GNU GPL for details.\n");
128    SerialOutputString("This is free software, and you are welcome "
129        "to redistribute it\n");
130    SerialOutputString("under certain conditions; "
131        "read the GNU GPL for details.\n");

```

```

125     get_memory_map();
126     /*
127      * elf 섹션에 대한 정보를 시리얼로 출력한다.
128      */
129     print_elf_sections();
130 #ifdef BLOB_DEBUG
131 {
132     register u32 stackptr;
133     SerialOutputString("Current stack pointer: 0x");
134     asm("mov %0, sp": "=r" (stackptr));
135     SerialOutputHex(stackptr);
136     serial_write("\n");
137 }
138 #endif
139
140     /* 파라미터 블록에 있는 모든 태그들을 해석한다.
141      * Parse all the tags in the paramater block */
142 #ifdef PARAM_START
143     //parse_ptags((void *) PARAM_START, &conf);
144 #endif
145
146     /* do_reload() 함수 호출로 "blob", "kernel", "ramdisk" 의 인자를 주어 플래
147      * 쉬로 부터 램으로 커널과 램디스크를 로드한다. 램디스크를 사용한다면
148      * 아래 주석처리 되어 있는 부분을 반드시 해제시켜야 한다.
149      * - [4.3.2. 호출된 중요 루틴들]중 D.do_reload() 함수 참조 -
150      */
151     /* Load kernel and ramdisk from flash to RAM */
152     do_reload("blob");
153     do_reload("kernel");
154     /* ←-없앰
155     if(blob_status.load_ramdisk)
156         do_reload("ramdisk");
157     */ ←-없앰
158
159     /* 지정한 시간(여기서는 10초)동안의 키입력을 기다린다.
160      * 이는 main()함수의 맨처음 blob_status.boot_delay에 10을 주었었고, 실제
161      * 대기는 $(BLOBTOP)/src/lib/serial.c에 있는 SerialInputBlock()함수에서 1초
162      * 동안 키입력을 기다린후 리턴하므로 결국 총 10초간 대기하게 된다
163      */
164     SerialOutputString("Autoboot in progress, press any key to stop ");
165     for(i = 0; i < blob_status.boot_delay; i++) {
166         serial_write('.');
167
168         retval = SerialInputBlock(commandline, 1, 1);
169
170         if(retval > 0)
171             break;
172     }
173
174     /* retval이 0, 즉 키가 눌리지 않았다면 Auto Boot를 실행한다.
175      * - [4.3.2. 호출된 중요 루틴들]중 F.parse_command() 함수 참조 -
176      * - [4.3.2. 호출된 중요 루틴들]중 G.Auto Boot 과정 참조 -
177      */
178     if(retval == 0) {
179         commandline[0] = '\0';
180         parse_command("boot");
181     }

```

168
169

■ 키가 눌렀다면, 메뉴루틴으로 들어간다.

```
170 SerialOutputString("\nAutoboot aborted\n");
171 SerialOutputString("Type \"help\" to get a list of commands\n");
172
173 DisplayPrompt(NULL);
174 numRead = 0;
175 done = 0;
176
```

■ 기간내에 키가 눌렀다면 키를 입력받아 메뉴항목과 비교하여 해당 루틴을 실행한다.

위의 코드는 for 루프를 돌면서 GetCommand ()함수를 통해서 commandline에 입력되는 명령어를 읽고, 이를 위에서 본 parse_command()함수에 인수로 주어 해당 루틴을 수행한다. GetCommand()함수는 lib/command.c에 코드가 존재하고 있으며, 하는 일은 일정 시간동안 혹은 "신호가 들어올 동안 키 입력을 받는 역할을 한다. 받은 수를 리턴한다.

```
178 for(;;) {
179     char c;
180     char ready;
181
182     /* wait 10 minutes for a command */
183     //numRead = GetCommand(commandline,
MAX_COMMANDLINE_LENGTH, 600);
184
185 #if ( defined(LUBBOCK) )
186     eth_rx();
187 #endif
188
189     ready = serial_poll();
190     if ( ready == 1 ) {
191
192         c = serial_read();
193
194         if(c < 0) {
195             serial_write("\n");
196             continue;
197         }
198         /* '\r', '\n'신호가 들어오면 done을 설정하여 하나의 명령이 들
어왔
음을 표시한다.*/
199         if((c == '\r') || (c == '\n')) {
200             commandline[numRead++] = '\0';
201             done = 1;
202
203             serial_write("\n");
204         } else if(c == '\b') { /*백스페이스에 대한 처리를 한다.*/
205             if(numRead > 0) {
206                 numRead--;
207                 SerialOutputString("\b\b");
208             }
209         } else { /*들어온 문자를 버퍼에 저장한다.*/
210             commandline[numRead++] = c;
211
212             serial_write(c);

```

```

213         }
214     }
215
216
217     if(done) {
        done이 설정되어 하나의 명령문자열이 들어오면
        parse_command()함수를 호출하여 이에 대한 처리를 한
        다.
        - [4.3.2. 호출된 중요 루틴들]중 F.parse_command() 함수
        참조 -
218         if((retval = parse_command(commandline)) < 0 )
219             prnterror(retval, NULL);
220
221         /* reset count */
222         done = 0;
223         numRead = 0;
224
225         DisplayPrompt(NULL);
226     }
227 }
228
229 return 0;
230 } /* main */

```

4.3.2. 호출된 중요 루틴들

이 곳에서는 위의 main()함수에서 호출하였던 함수들 중 중요한 함수들 및 중요 구조체들을 선택해서 살펴보도록 한다.

4.3.2.1. init_subsystems() 함수

Main()함수는 맨 처음 init_subsystems() 함수를 호출함으로써 등록된 모든 초기화 함수를 실행한다. 이 init_subsystems()함수 소스에서는 __initlist_start와 __initlist_end영역 사이에 존재하는 모든 초기화 함수들을 실행시킨다. 이 영역은 rest-ld-script화일에 아래와 같이 정의되어 있다.

\$(BLOBTOP)/src/blob/rest-ld-script 화일중

```

. = ALIGN(4);
    .initlist : {
        __initlist_start = .;
        *(.initlist)
        __initlist_end = .;
    }

```

이 초기화 영역에 초기화 함수를 등록하기 위해서는 include/blob/init.h 에 아래 매크로를 사용한다.

\$(BLOBTOP)/include/blob/init.h 소스중


```
#define __init __attribute__((unused, __section__(".initlist")))

#define __initlist(fn, lvl) \
static initlist_t __init_##fn __init = { \
    magic: INIT_MAGIC, \
    callback: fn, \
    level: lvl } }
```

이 매크로의 사용은 `__initlist(fn, lvl)` 형식으로 초기화가 필요한 각 소스파일에
`“__initlist(init_flash_driver, INIT_LEVEL_DRIVER_SELECTION);”`
 와 같이 선언하여 `callback` 함수와 `level` 값을 선언한다. 이렇게 선언함으로써 해당
 함수는 `.initlist` 섹션에 등록되어 진다.

아래소스는 `init_subsystems()` 소스를 보인 것이다..

`$(BLOBTOP)/src/lib/init.c` 소스중

이 함수는 `init_subsystems()` 함수에서 호출되며, `initlist` 영역을 루프를 돌면서 등록된 것들
 을 실행한다.

```
static void call_funcs(initlist_t *start, initlist_t *end, u32 magic, int level)
{
    initlist_t *item;

    for(item = start; item != end; item++)
    {
        /*우선 INIT_MAGIC의 값이 제위치에 존재하는지를 확인한다. 값이 틀리다면 에
        러가 있는 것으로 분명히 위의 매크로에서 INIT_MAGIC을 magic영역에 두었었
        다.*/
        if(item->magic != magic) {
            printerror(EMAGIC, NULL);

#ifdef BLOB_DEBUG
            printerrprefix();
            SerialOutputString("Address = 0x");
            SerialOutputHex((u32)item);
            serial_write("");
#endif
            return;
        }

        /*에러가 없고 레벨값이 같다면 등록된 해당 초기화함수를 실행한다.*/
        if(item->level == level) {
            /* call function */
            item->callback();
        }
    }
}

init_subsystems에서는 단순히 INIT_LEVEL 크기에 따른 우선순위로 call_funcs를 호출함
으로써 초기화를 이룬다.
void init_subsystems(void)
{
    int i;
```

```

/* init섹션에 있는 모든 초기화 루틴들을 레벨값 순서로 실행한다. */
for(i = INIT_LEVEL_MIN; i <= INIT_LEVEL_MAX; i++)
    call_funcs((initlist_t *)&__initlist_start, (initlist_t *)&__initlist_end,
               INIT_MAGIC, i);
}

```

4.3.2.2. Blob 구조체 (blob_status_t)

blob_status_t 구조체의 정의는 main.h에 정의되어 있으며 아래와 같다.

아래 구조체 필드중 block_source_t 구조체는 두 개의 필드가 있으며, Flash에서 읽을 것인지, 다운로드 받을것인지에 대한 값을 enum 타입으로 정의한다.

\$(BLOBTOP)/include/blob/main.h 소스중

```

typedef enum {
    fromFlash = 0,
    fromDownload = 1
} block_source_t;

typedef struct {
    int kernelSize;                /*커널의 크기*/
    block_source_t kernelType;     /*커널을 어디서 가져오는 지를 정의한다.*/
    u32 kernel_md5_digest[4];

    int paramSize;
    block_source_t paramType;
    u32 param_md5_digest[4];

    int ramdiskSize;               /*ram Disk 크기*/
    block_source_t ramdiskType;    /*Ram disk를 어디서 가져오는지를 정의한다.*/
    u32 ramdisk_md5_digest[4];

    int blobSize;                  /*BLOB 자체의 크기*/
    block_source_t blobType;       /*BLOB를 어디서 가져오는 지를 정의한다.*/
    u32 blob_md5_digest[4];

    serial_baud_t downloadSpeed; /*DownLoading Speed를 정의한다.*/
    serial_baud_t terminalSpeed; /*터미널의 baudrate를정한다.*/

    int load_ramdisk;
    int boot_delay;

    char cmdline[COMMAND_LINE_SIZE];
} blob_status_t;

```

4.3.2.3. serial_init() 함수

serial_init()함수에 Baudrate를 인수로 주어 호출하여 시리얼 포트를 초기화 시키게 된다. serial_init()함수는 아래와 같이 /lib/serial.c화일에 존재한다.

\$(BLOBTOP)/src/lib/serial.c 소스중

```
int serial_init(serial_baud_t baudrate)
{
#ifdef BLOB_DEBUG
    if(serial_driver == NULL) {
        /* serial_driver는 초기화 함수실행 중에 할당하게 되며, 이것이 Null이라면 리턴한
        다. */
        return -ERANGE;
    }
#endif
    /*해당 초기화를 실행한다*/
    return serial_driver->init(baudrate);
}
```

위 코드에서는 단지 serial_driver구조체에 할당된 init()함수로 baudrate를 전달하는 역할만 하며 시리얼 핸들러 함수 포인터를 담고 있는 이 serial_driver구조체는 src/blob/kelb.c파일에서 아래와 같이 pxa_serail_driver구조체로 할당하고 있다.

\$(BLOBTOP)/src/blob/lubbock.c 소스중

```
static void kelb_init_hardware(void)
{
    unsigned long i, addr, end, stack_end;

    /* select serial driver */
    serial_driver = &pxa_serial_driver;
    /*기타 특정 하드웨어에 대한 초기화를 한다.*/
    ....
}
__initlist(kelb_init_hardware, INIT_LEVEL_DRIVER_SELECTION);
```

위에서 보면 __initlist매크로를 사용하여 kelb_init_hardware(void)함수를 초기화 루틴에 넣고 있으며 초기화 루틴 실행중에 이 함수에 의해서 pxa_seraial_driver 포인터를 serial_driver구조체의 포인터에 넣게된다. pxa_serial_driver구조체는 아래와 같이 선언되어 있다.

\$(BLOBTOP)/src/lib/serial-pxa.c 소스중

```
/* export serial driver */
serial_driver_t pxa_serial_driver = {
    init:          pxa_serial_init,
    read:          pxa_serial_read,
    write:         pxa_serial_write,
    poll:          pxa_serial_poll,
    flush_input:   pxa_serial_flush_input,
    flush_output:  pxa_serial_flush_output
};
```

위의 구조체에서는, init, read, write등의 시리얼 포트의 사용에 관련된 모든 함수

들의 포인터가 할당되게 되며, 결국 여기서 할당된 함수포인터들이 SerialOutputString()등의 시리얼과 관련된 루틴들이 호출되고 있음을 볼 수 있을 것이다.

SerialOutputString()의 함수들은 \$(BLOBTOP)/src/lib/serial.c화일에 선언되어 있으며, 이들 함수들에 대해서는 쉽게 이해할 수 있는 내용으로 각자 보도록 한다.

결국 시리얼 포트의 초기화는 pxa_serial_init()함수에 의하여 수행되는데 소스는 아래와 같으며, 이 함수에 대하여서만 보도록 하겠다. 아래에서 FFUART를 사용하지 않을 경우 FFxxx로 되어 있는 부분을 STUART, HWUART등으로 바꾸어 주면 된다. 바꾸어 주기전에 해당 GPIO핀에 대한 부가기능설정이 정확히 설정되어 있어야 함을 유념하자.

\$(BLOBTOP)/src/lib/serial-pxa.c 소스중

```
#ifndef HAVE_CONFIG_H
# include <blob/config.h>
#endif
```

```
#include <blob/pxa.h>
#include <blob/errno.h>
#include <blob/serial.h>
#include <blob/types.h>
```

사용할 시리얼 포트의 레지스터를 설정하는 것으로 사용하려는 시리얼 포트를 이곳에서 수정하면 된다.

```
#define UART_RBR          FFRBR
#define UART_THRFFTHR
#define UART_LCRFFLCR
#define UART_LSR          FFLSR
#define UART_IER FFIER
#define UART_FCRFFFCR
#define UART_DLL FFDLL
#define UART_DLHFFDLH
#define UART_MCR          FFMCR
```

Baud Rate 설정을 한다.

```
#define XTAL    14745600
static int pxa_serial_change_speed(int baud) {
    int divisor = XTAL / (baud<<4);

    UART_LCR |= LCR_DLAB;

    UART_DLL = (divisor & 0xFF);
    UART_DLH = (divisor >> 8);

    UART_LCR &= ~LCR_DLAB;

    return 0;
}
```

시리얼 입력큐를 비우는 작업을 한다.

```
static int pxa_serial_flush_input(void)
{
    volatile u32 tmp;

    /* keep on reading as long as the receiver is not empty */
    while( UART_LSR & LSR_DR ) {
        if( UART_LSR & ( LSR_PE | LSR_FE | LSR_BI ) )
            return -ESERIAL;

        tmp = UART_RBR;
    }

    return 0;
}
```

시리얼 출력큐를 비우는 작업을 한다.

```
static int pxa_serial_flush_output(void)
{
    /* wait until the transmitter is ready to transmit*/
    while( !(UART_LSR & LSR_TDRQ) );

    return 0;
}
```

시리얼 출력을 위하여서는 우선 해당 GPIO의 부가기능 설정을 해야 하며, 이를 위하여 set_GPIO_mode()함수를 추가하였다.

```
static void set_GPIO_mode(int gpio_mode)
{
    int gpio=gpio_mode & GPIO_MD_MASK_NR;
    int fn=(gpio_mode & GPIO_MD_MASK_FN)>>8;
    int gafr;

    if(gpio_mode & GPIO_MD_MASK_DIR)
        GPDR(gpio) |=GPIO_bit(gpio);
    else
        GPDR(gpio)&=~GPIO_bit(gpio);
    gafr=GAFR(gpio)&~(0x3<<(((gpio)&0xf)*2));
    GAFR(gpio)=gafr|(fn<<(((gpio)&0xf)*2));
}
```

요구된 Baud Rate에 대한 설정을 한다.

성공하면 0을 실패하면 음수를 리턴한다. 보레이트 계산은 아래와같이 하며

```
*
*
*          147.7456 MHZ
* BaudRate = -----
*          16 x Divisor
*
```

따라서, divisor 가 24라면 , 보레이트는 38400 bps가 된다.

```
static int pxa_serial_init(serial_baud_t baud)
{
    CKEN |= CKEN6_FFUART;
    /*GPIO 핀 설정에 대해 한다.*/
    set_GPIO_mode(GPIO34_FFRXD_MD);
    set_GPIO_mode(GPIO39_FFTXD_MD);
    /* set the port to sensible defaults (no break, no interrupts,
    * no parity, 8 databits, 1 stopbit, transmitter and receiver
```

```

    * enabled)
    */
    UART_LCR = 0x3;

    /* assert DTR & RTS to avoid problems of hardware handshake
    * with serial terminals
    */
    UART_MCR = MCR_DTR | MCR_RTS ;

    /* disable FIFO */
    UART_FCR = FCR_TRFIFOE;

    /* enable UART */
    UART_IER = IER_UUE;

    pxa_serial_change_speed(baud);

    pxa_serial_flush_output();

    return 0;
}

```

4.3.2.4. get_memory_map() 함수

이 함수는 시스템에 존재하는 메모리를 1Mbyte 단위의 블록으로 관리하기 위하여 `memory_map[]` 배열에 `testram()` 루틴을 사용하여 램이 존재하는 지 확인한 후 해당 블록에 메모리가 존재한다는 표식을 하고, 시스템의 메모리정보를 시리얼로 출력하는 작업을 한다.

\$(BLOBTOP)/src/blob/memory.c 소스중

```

memory_area_t memory_map[NUM_MEM_AREAS];
void get_memory_map(void)
{
    u32 addr;
    int i;
    메모리 맵의 모든 영역을 사용하지 않는 것으로 초기화
    for(i = 0; i < NUM_MEM_AREAS; i++)
        memory_map[i].used = 0;
    각 메모리 영역의 처음부분에 0을 써넣는다.
    for(addr = MEMORY_START; addr < MEMORY_END; addr +=
TEST_BLOCK_SIZE)
        *(u32 *)addr = 0;
}

```

메모리의 시작(MEMORY_START)과 메모리의 마지막주소(MEMORY_END) 사이의 모든 메모리 영역에 대해서 다시 TEST_BLOCK_SIZE(1MBYTEs단위) 크기로 루프를 돌면서 실제로 메모리가 있는지 확인하는 작업을 한다.

여기서 쓰인 testram함수는 testmem.S에 들어 있는 함수가 아닌 testmem2.S에 들어 있는 함수로 레지스터들을 먼저 저장하고 사용한 후 복원시킨다.

testram()함수의 결과가 0이라면 메모리가 존재하는 경우이고, 이 경우에 만약 앞에서 각 영역의 맨처음에 설정한 0이 아니라면, 이 메모리가 alias되어서 사용되고 있는것 일수도 있으므로 이때는 memory_map[]의 해당 block에 대한 used필드가 1로 설정된 경우에만 블록의 수를 증가시켜주기 위해서 i를 증가시키고 다음 루프로 진행한다.

만약 0이라면, 현재의 주소값 (addr)을 그대로 addr변수가 가르키는 곳에 적어주어서, alias인지를 판단할 수 있도록 한 후에 현재의 block이 사용중이 아니라면 시작주소에는 addr을 기입하고, 크기는 1Mbytes(=TEST_BLOCK_SIZE), used 필드에는 1로 준다. 사용중이라면 단지 크기만을 증가시켜 줌으로써 이전 설정한 block의 일부로 만들어 준다.(TEST_BLOCK_SIZE를 더해줌).

그 다음 만약 BLOB_DEBUG가 define 되어 있다면 디버깅정보를 표시하기 위하여 각각의 memory_map[] element에 대해서 loop를 돌면서 메모리의 시작과 길이, 그리고 크기를 시리얼로 출력한다.

testram()함수의 결과가 0이 아니라면 RAM이 존재하지 않는 것이므로 memory_map[]에 해당 블록의 used 필드가 1인 경우에만 블록의 개수를 나타내는 변수 인 i를 증가시켜 주고, 그렇지 않다면 다음 loop로 진행한다.

```
i = 0;
for(addr = MEMORY_START; addr < MEMORY_END; addr +=
    TEST_BLOCK_SIZE) {
    /* testram함수로 해당 영역이 램임을 확인한다. */
    if(testram(addr) == 0) {
        if(* (u32 *)addr != 0) { /* alias 라면? */
#ifdef BLOB_DEBUG
            SerialOutputString("Detected alias at 0x");
            SerialOutputHex(addr);
            SerialOutputString(", aliased from 0x");
            SerialOutputHex(* (u32 *)addr);
            serial_write("");
#endif
            /*현재블록이 alias이고 사용중이라면 i를 증가시키고 다음 루프로 돈다.*/
            if(memory_map[i].used)
                i++;
            continue;
        }

        /* alias가 아니라면 현재 주소값을 쓴다. */
        * (u32 *)addr = addr;
#ifdef BLOB_DEBUG
        SerialOutputString("Detected memory at 0x");
        SerialOutputHex(addr);
        serial_write("");
#endif
#endif
}
```

```

/* 새로운 블록이라면, 현재주소를 start에 써주고 블록의 크
기를 len 필드에 써주고, 마지막으로 사용중이라는 표시를 하
기위해 used에 1을 써준다. */
if(memory_map[i].used == 0) {
    memory_map[i].start = addr;
    memory_map[i].len = TEST_BLOCK_SIZE;
    memory_map[i].used = 1;
} else {
/*새로운 블록이 아니라면, 사용중인 블록의 길이를 늘려주기
만 한다.*/
    memory_map[i].len += TEST_BLOCK_SIZE;
}
} else { // if(testram(addr) != 0)
/* 램이 없다고, 현재 블록이 사용중이라면, 새로운 블록할당을 위해 1
를 증가시킨다. */
    if(memory_map[i].used == 1)
        i++;
}
}
위에서 찾은 메모리 블록들에 대한 정보를 시리얼로 출력한다.
SerialOutputString("Memory map:");
for(i = 0; i < NUM_MEM_AREAS; i++) {
    if(memory_map[i].used) {
        SerialOutputString(" 0x");
        SerialOutputHex(memory_map[i].len);
        SerialOutputString(" @ 0x");
        SerialOutputHex(memory_map[i].start);
        SerialOutputString(" (");
        SerialOutputDec(memory_map[i].len / (1024 * 1024));
        SerialOutputString(" MB)");
    }
}
}

```

4.3.2.5. do_reload()함수

Main함수에서 do_reload()함수에 "blob", "kernel", "ramdisk"의 인수를 주어 Flash에 있는 코드들을 램으로 옮기는 작업을 하게 됨을 보았다. 여기서는 그 과정에 관하여 보도록 할 것이다. do_reload()함수의 코드는 아래와 같다.

\$(BLOBTOP)/src/blob/main.c 소스중

```

플래쉬의 내용을 램으로 옮기는 작업을 하며, 각 변수의 정의는 include/blob/arch/kelb.h
에 정의하여 두었다.
여기서의 RAMDISK 위치, KERNEL위치는 커널설정에서 세팅하는 사항과 동일해야 한
다. 이 부분에 대해서는 커널 초기화소스를 보면서 보도록 할 것이다.
static int do_reload(char *what)
{
    u32 *dst = 0;
    u32 *src = 0;
    int numWords;

```


■ 넘어온 인수가 "blob" 일 경우 소스주소, 타겟주소, 복사할 워드수를 설정한다.

```
if(strncmp(what, "blob", 5) == 0) {
    dst = (u32 *)BLOB_RAM_BASE;
    src = (u32 *)BLOB_FLASH_BASE;
    numWords = BLOB_FLASH_LEN / 4;
    blob_status.blobSize = 0;
    blob_status.blobType = fromFlash;
    SerialOutputString("Loading blob from flash ");
}
```

#ifdef PARAM_START

■ 넘어온 인수가 "param" 일 경우 소스주소, 타겟주소, 복사할 워드수를 설정한다.

```
else if(strncmp(what, "param", 6) == 0) {
    dst = (u32 *)PARAM_RAM_BASE;
    src = (u32 *)PARAM_FLASH_BASE;
    numWords = PARAM_FLASH_LEN / 4;
    blob_status.paramSize = 0;
    blob_status.paramType = fromFlash;
    SerialOutputString("Loading paramater block from flash ");
}
```

#endif

■ 넘어온 인수가 "kernel" 일 경우 소스주소, 타겟주소, 복사할 워드수를 설정한다.

```
else if(strncmp(what, "kernel", 7) == 0) {
    dst = (u32 *)KERNEL_RAM_BASE;
    src = (u32 *)KERNEL_RAM_BASE;
    numWords = KERNEL_FLASH_LEN / 4;
    blob_status.kernelSize = 0;
    blob_status.kernelType = fromFlash;
    SerialOutputString("Loading kernel from flash ");
}
```

■ 넘어온 인수가 "ramdisk" 일 경우 소스주소, 타겟주소, 복사할 워드수를 설정한다.

```
else if(strncmp(what, "ramdisk", 8) == 0) {
    dst = (u32 *)RAMDISK_RAM_BASE;
    src = (u32 *)RAMDISK_FLASH_BASE;
    numWords = RAMDISK_FLASH_LEN / 4;
    blob_status.ramdiskSize = 0;
    blob_status.ramdiskType = fromFlash;
    SerialOutputString("Loading ramdisk from flash ");
}
else {
    prnterror(EINVAL, what);
    return 0;
}
```

■ MyMemCpy함수로 소스주소에서 타겟메모리로 내용을 복사한다.

```
MyMemCpy(dst, src, numWords);
SerialOutputString(" done");
```

```
return 0;
```

```
}
```

4.3.2.6. parse_command()함수

이 parse_command()는 키보드로 들어온 문자열을 받아 해석하여 해당하는 명령

콜백 함수를 인자가 있다면 인자와 함께 호출하는 일을 한다.

\$(BLOBTOP)/src/lib/command.c 소스중

```
int parse_command(char *cmdline)
{
    commandlist_t *cmd;
    int argc, num_commands, len;
    char *argv[MAX_ARGS];
    ■ cmdline 문자열로부터 명령행과 인수의 갯수를 분석한다
    parse_args(cmdline, &argc, argv);

    /* 인수가 없다면 리턴한다. */
    if(argc == 0)
        return 0;
    ■ 명령행과 같은 것이 있는지 확인한다.
    num_commands = get_num_command_matches(argv[0]);

    /* num_commands가 0보다 작다면 에러가 있는 것이다 */
    if(num_commands < 0)
        return num_commands;

    /* 일치하는 명령의 갯수가 0이라면 에러리턴한다 */
    if(num_commands == 0)
        return -ECOMMAND;

    /* 1보다 많다면 에러리턴한다. */
    if(num_commands > 1)
        return -EAMBIGCMD;

    len = strlen(argv[0]);

    /* single command, go for it */
    ■ parse_command() 함수에서는 등록된 commands리스트를 검색하여 인수로
    받은 argv[0]스트링과 같은 항목의 callback함수를 실행시킨다.
    - [4.3.2. 호출된 중요 루틴들]중 F.1. commandlist 구조체 참조 -
    for(cmd = commands; cmd != NULL; cmd = cmd->next) {
        /*명령 MAGIC번호가 일치하지 않으면 에러처리 */
        if(cmd->magic != COMMAND_MAGIC) {
#ifdef BLOB_DEBUG
            printerrprefix();
            SerialOutputString(__FUNCTION__ "(): Address = 0x");
            SerialOutputHex((u32)cmd);
            serial_write("");
#endif

            return -EMAGIC;
        }
        /*argv[0] 스트링이 commands 리스트중에 같은 것이 있다면 넘어온 인자
        들을 인수로 주어 콜백함수를 실행하고 리턴한다. */
        if(strncmp(cmd->name, argv[0], len) == 0) {
            return cmd->callback(argc, argv);
        }
    }
    return -ECOMMAND;
}
```

A. commandlist 구조체

위의 `parse_command()` 함수에서 명령을 찾기 위하여 `commands`영역을 탐색하는 것을 보았다. 이 `Commands` 리스트에 등록하는 것은 `__commandlist` 매크로를 사용하며, 매크로에 대한 소스는 `include/blob/command.h`에 아래와 같이 나온다.

`$(BLOBTOP)/include/blob/command.h` 소스중

```
typedef int(*commandfunc_t)(int, char *[]);

typedef struct commandlist {
    u32 magic;
    char *name;
    char *help;
    commandfunc_t callback;
    struct commandlist *next;
} commandlist_t;

#define __command __attribute__((unused, __section__(".commandlist")))

#define __commandlist(fn, nm, hlp) \
parstatic commandlist_t __command_##fn __command = \
\par    magic:    COMMAND_MAGIC, \par    name:    nm, \par    help:    hlp, \
\par    callback: fn
```

위 코드에서 보면 매크로에 전달된 인수들을 통해서 `magic`, `name`, `help`, `callback`의 순서로 `.commandlist` 섹션에 놓는 것을 볼 수 있다. 이때 `magic`에는 `COMMAND_MAGIC`상수 값이 들어 가며, 이 상수값은 명령문자열을 비교하기 전에 먼저 확인한다. `commandlist`섹션은 `rest-ld-script.in`소스에서 확인할 수 있다.

`$(BLOBTOP)/src/blob/rest-ld-script.in`

```
. = ALIGN(4);
    .commandlist :
        __commandlist_start = .;
        *(.commandlist)
        __commandlist_end = .;
```

`commands` 리스트에 대한 것은 `lib/command.c`화일에 보면 `comandlist_t` 구조체 타입으로 `commands`를 선언하고 있으며, `__initlist`매크로를 사용하여 `init_commands()`함수를 초기화 함수로 등록하고, 이 함수에서 `commands`변수에 `__commandlist_start`를 주어 초기화를 한다. 그 후에 `__commandlist_start`부터 `__commandlist_end` 영역안에 존재하는 `commandlist_t` 타입들에 대한 링크된 리스트를 만든다.

`$(BLOBTOP)/src/lib/command.c` 소스중

```

commandlist_t *commands;          /* 리스트의 맨처음 부분 변수 */
static void init_commands(void)
{
    commandlist_t *lastcommand;
    commandlist_t *cmd, *next_cmd;

    /*commands에 리스트 영역의 헤더를 할당한다*/
    commands = (commandlist_t *) &__commandlist_start;
    /*lastcommand에 리스트 영역의 마지막을 할당한다*/
    lastcommand = (commandlist_t *) &__commandlist_end;

    /* __commandlist 매크로를 사용하여 등록된 command 들의 링크드 리스트를
    만든다 */
    cmd = next_cmd = commands;
    next_cmd++;

    while(next_cmd < lastcommand) {
        cmd->next = next_cmd;
        cmd++;
        next_cmd++;
    }
}
/* 초기화 루틴에 등록한다 */
__initlist(init_commands, INIT_LEVEL_OTHER_STUFF);

```

4.3.2.7. Auto Boot 과정

여기에서는 키입력이 없었을 때 수행되는 `parse_command()` 함수에 "boot"인수를 주었을 경우의 명령에 대한 콜백함수인 `boot_linux()` 함수를 보도록 한다.

이제까지 `commands` 리스트에 명령과 콜백함수들을 등록하는 매크로에 대하여 보았다. 여기서는 `commandlist` 영역에 명령을 등록하는 방법과 사용방법에 대하여, "boot"인자에 대한 핸들러 코드를 보면서 이해하도록 하자.

명령을 등록하는 방법은 간단하다. 우선 핸들러함수를 정의/구현하고, `__commandlist` 매크로를 사용하여 `__commandlist` 영역에 넣어주면 된다.

"boot"인자를 주어 자동부팅을 실행함으로 이것에 대하여 보도록 하자.

\$(BLOBTOP)/src/blob/linux.c 소스중

```

■ __commandlist 매크로에 핸들러 포인터, 메뉴에 표시될 명령이름, 도움말문자열 인
자를 주어 commandlist 영역에 추가한다.
키입력을 10초동안 기다리는 루틴에서 놀리지 않았을 경우 parse_command("boot")를
통하여 부팅을 하게 되며, 이 parse_command("boot")를 호출하였을 경우 여기서 설정한
핸들러가 실행하게 된다.
__commandlist(boot_linux, "boot", boothelp);
■ "boot"명령에 대한 핸들러
static int boot_linux(int argc, char *argv[])
{

```

■ 재배치한 커널코드의 맨처음 실행주소를 함수 포인터를 써서 호출 설정한다. 여기에 인수로 `int zero`와 `arch`를 두고 있음을 주목하자. 이 것이 뜻하는 것은 `r0`에는 `zero`값이, `r1`에는 `arch`값이 들어 가서 커널코드로 분기될 것이라는 것이다.

```
void (*theKernel)(int zero, int arch) =
    (void (*)(int, int)) KERNEL_RAM_BASE;
```

■ 각 태그들을 설정하는데 여기서는 커널로 넘겨줄 정보가 있다면, 이들 태그 영역을 사용하여 넘겨주면 된다. 주석처리 되어 있다.

- [4.3.2. 호출된 중요 루틴들]중 G.1. tag 정보 참조 -
- [4.3.2. 호출된 중요 루틴들]중 G.2. `setup_start_tag()`함수 참조 -
- [4.3.2. 호출된 중요 루틴들]중 G.3. `setup_memory_tags()`함수 참조 -
- [4.3.2. 호출된 중요 루틴들]중 G.4. `setup_command_tag()`함수 참조 -
- [4.3.2. 호출된 중요 루틴들]중 G.5. `setup_initrd_tag()`함수 참조 -
- [4.3.2. 호출된 중요 루틴들]중 G.6. `setup_ramdisk_tag()`함수 참조 -
- [4.3.2. 호출된 중요 루틴들]중 G.7. `setup_end_tag()`함수 참조 -

```
/*
    setup_start_tag();
    setup_memory_tags();
    setup_commandline_tag(argc, argv);
    setup_initrd_tag();
    setup_ramdisk_tag();
    setup_end_tag();
*/
```

■ "kernel..."문자를 시리얼로 출력한후 시리얼 출력버퍼를 비운다.

```
SerialOutputString("\nStarting kernel ...\n\n");
serial_flush_output();
```

■ `exit_subsystems()`함수를 호출하여 커널부팅을 하기전에 `__exitlist`영역에 등록된 함수들을 실행한다. 이는 전에 보았던 `__initlist`영역의 실행과 비슷하다.

- [4.3.2. 호출된 중요 루틴들]중 G.8. `exit_subsystems()`함수 참조 -

```
exit_subsystems();
```

■ 위에서 설정한 `theKernel()`함수를 호출함으로써 `KERNEL_RAM_BASE`위치로의 분기가 이루어 지며, `r0=0`, `r1=ARCH_NUMBER` 값이 들어가게 된다.

```
theKernel(0, ARCH_NUMBER);
```

```
/* 커널코드로 부터 돌아올 일은 없으므로 아래 코드는 실행되서는 않된다*/
```

```
SerialOutputString("Hey, the kernel returned! This should not happen.");
return 0;
```

```
}
```

관련정보

커널소스 `r0`에는 `00i`, `r1`에는 `ARCH_NUMBER`가 들어가 커널 실행코드로 분기된다. 넘어간 후에는 압축된 커널코드를 실행시키기 위하여 이를 푸는 실행코드가 될 것이다. 만약 여기서 넘어간 인자 `ARCH_NUMBER`가 맞지 않는다면 부팅도중 정지하고 있는 상태가 될 것이다. 여기에 대해서는 <부팅에서 프래프트까지>를 살펴보기 바란다.

넘겨주는 `architecture` 번호에 대한 정의는 `linux.h`에 아래와 같이 되어 있다. 같은 것을 커널 소스의 `~/arch/arm/tools/mach_types`에서 찾을 수 있다.

```
$(BLOBTOP)/include/blob/linux.h 소스중
```

```
#if defined ASSABET
# define ARCH_NUMBER (25)
#elif defined BADGE4
# define ARCH_NUMBER (138)
```

```
#elif defined BRUTUS
# define ARCH_NUMBER (16)
.....
#else
#warning "FIXME: Calling the kernel with a generic SA1100 architecture code. YMMV!"
#define ARCH_NUMBER (18)
#endif
```

parse_command()함수는 위에서 보았으며, 여기에 __commandlist 매크로를 써서 등록한 함수들의 종류는 아래와 같다.

```
☞ $(BLOBTOP)/src/lib/command.c      :
    __commandlist(help, "help", helphelp);
☞ $(BLOBTOP)/src/lib/commands.c    :
    __commandlist(reset_terminal, "reset", resethelp);
    __commandlist(reboot, "reboot", reboothelp);
☞ $(BLOBTOP)/src/blob/linux.c      :
    __commandlist(boot_linux, "boot", boothelp);
☞ $(BLOBTOP)/src/blob/main.c       :
    __commandlist(Download, "download", downloadhelp);
    __commandlist(xdownload, "xdownload", xdownloadhelp);
    __commandlist(Flash, "flash", flashhelp);
    __commandlist(SetDownloadSpeed, "speed", speedhelp);
    __commandlist(PrintStatus, "status", statushelp);
    __commandlist(Reload, "reload", reloadhelp);
☞ $(BLOBTOP)/src/blob/reboot.c     :
    __commandlist(reblob, "reblob", reblobhelp);
☞ $(BLOBTOP)/src/blob/system3.c    :
    __commandlist( cmd_flash_write, "fwrite", flashwritehelp );
    __commandlist( cmd_download_file, "dfile", downloadhelp );
    __commandlist( cmd_flash_erase, "ferase", flasherashelp );
```

등이 있다. 이들에 대한 코드는 보면 알만한 내용들로 각자 보도록 한다. 플래쉬에 대한 내용은 JTAG 편을 참조하라.

☞ JTAG를 사용하여 플래쉬에 일일이 쓰는 것이 귀찮게 느껴 진다면, 환경설정에서 보았던 TFTP를 이용하여 플래쉬에 쓰는 방법을 쓰도록 하라. 이 TFTP를 사용하기 위해서는 보드에 TFTP에 대한 코딩을 해주어야 하며 이 메뉴리스트에 위 처럼 추가해 주어야 한다. 다행스럽게도 \$(BLOBTOP)/src/blob/net.c와 net.h파일에서 tftp, arp등에 대한 프로토콜을 지원하고 있으며, LAN91C111칩에 대한 드라이버소스를 smc91x.c와 smc91x.h 에서 제공하고 있으므로 이들을 응용하면 된다. 물론 인텔 플래쉬에 대한 드라이버도 intel16.c와 intel32.c 에서 제공하고 있다.

A. tag 정보

Tag정보를 설정하는 setup_xxx_tag()함수들은 주석처리 되어 있지만, 커널로 부트로더 내의 정보들을 넘기기 위한 방법이 필요할 수 있으므로 알아 두는 것도 좋을 듯 싶다. 이 tag 구조체를 포함하여 setup_XXX_tag에서 사용되는 상수 및 구조체는 리눅스 커널소스의 \$(KERNELTOP)/include/asm-arm/setup.h파일에 정의되어 있으며, 이 때문에 blob를 컴파일할 때 리눅스 소스 디렉토리를 정해주게 되는 것이다.

setup.h화일에는 또한 tag_size()매크로와 tag_next()매크로가 선언되어 있는데 tag_size 매크로는 tag_header의 크기와 해당하는 type의 크기를 합한 값을 4로 나눈 값을(워드크기) 돌려준다. tag_next()매크로는 다음 구조체의 위치를 돌려준다. 이들에 대한 코드는 리눅스 소스중 \$(KERNELTOP)/ include/asm-arm/setup.h 화일을 참조하라.

이 tag정보들을 사용하지 않을 수도 사용할 수도 있는데 사용할 경우 여기서 설정된 tag정보들은 커널 실행코드에서 ~/arch/arm/kernel/setup.c의 parse_tag_XXX()함수들에서 처리되며 여기서 수행하는 tag구조체 설정의 반대순으로 처리를 해 나가게 된다.

A.1. setup_start_tag()함수

이 함수는 BOOT_PARAMS(include/blob/arch/kelb.h에서 0xa0000100로 값을 주었다.)에 있는 tag구조체를 초기화 하는 역할을 한다. BOOT_PARAMS는 각 보드별 헤더화일 (여기서는 kelb.h)에 정의되어 있다.

소스중 나오는 ATAG_XXX 상수는 커널소스의 \$(KERNELTOP)/include/asm-arm/setup.h에 정의되어 있는데 여기서 쓰인 ATAG_CORE 0x54410001라는 값을 가진다. 크기는 tag_core를 인자로 주어 tag_size매크로에서 넘어온 크기 크기를 주고, flag=0, pagesize=0 ,rootdev=0을 준다. 그 다음 tag_next()를 사용하여 params 포인터가 다음 tag의 위치를 가리키도록 한다.

\$(BLOBTOP)/src/blob/linux.c 소스중

```
static void setup_start_tag(void)
{
    params = (struct tag *)BOOT_PARAMS;

    params->hdr.tag = ATAG_CORE;
    params->hdr.size = tag_size(tag_core);

    params->u.core.flags = 0;
    params->u.core.pagesize = 0;
    params->u.core.rootdev = 0;

    params = tag_next(params);
}
```

A.2. setup_memory_tags()함수

setup_memory_tags()함수는 위의 get_memory_map()함수에서 알아낸 현재 시스템의 메모리 정보를 설정한다. Tag의 tag필드에 메모리 태그라는 것을 나타내기 위하여 ATAG_MEM을 넣고 tag_size매크로를 사용하여 size에 크기를 넣은 후, 각각의 메모리 영역의 시작과 크기를 넣어준다. 그 다음 params 포인터에는 tag_next()매크로를 사용하여 다음 tag의 위치를 가리키도록 만든다. 위에서 찾아낸

메모리 영역의 개수만큼 ATAG_MEM을 tag항목으로 갖는 태그들이 만들어 진다.

\$(BLOBTOP)/src/blob/linux.c 소스중

```
static void setup_memory_tags(void)
{
    int i;

    for(i = 0; i < NUM_MEM_AREAS; i++) {
        if(memory_map[i].used) {
            params->hdr.tag = ATAG_MEM;
            params->hdr.size = tag_size(tag_mem32);

            params->u.mem.start = memory_map[i].start;
            params->u.mem.size = memory_map[i].len;

            params = tag_next(params);
        }
    }
    #if defined( NEPONSET )
        /* When neponset board is present dont report
        * the 2nd ram area to the kernel. This area
        * is on the neponset board and would require
        * NUMA support in the kernel.*/
        break;
    #endif
}
}
```

A.3. setup_commandline_tag()함수

command line에 해당하는 tag를 생성하게 된다. 우선 blob_status구조체의 필드 중 cmdline에 문자열이 들어 있다면 이 커맨드 문자열을 복사한 후에 넘어온 인자의 갯수가 2보다 많거나 같다면 스페이스 문자를 삽입하여 분리하고, 개수만큼 차례대로 복사해 넣는다.

이런 작업을 한 후에 결국 넘어온 커맨드 라인(문자열)이 있을 때만 tag를 등록한다. tag필드에 command line이라는 것을 표시하는 ATAG_CMDLINE을, size에는 구조체의 크기와 커맨드 라인의 문자열을 합하여 크기를 계산하여 설정한다. 마지막으로 params포인터가 다음 tag구조체를 가리키도록 한다.

\$(BLOBTOP)/src/blob/linux.c 소스중

```
static void setup_commandline_tag(int argc, char *argv[])
{
    char *p;
    int i;

    /* commandline 초기화*/
    params->u.cmdline.cmdline[0] = '\0';

    /* parameter block 으로 부터 blob_status구조체의 commandline을
```



```

COMMAND_LINE_SIZE(1024) 크기만큼 복사한다. */
if(blob_status.cmdline[0] != '\0')
    strcpy(params->u.cmdline.cmdline, blob_status.cmdline,
           COMMAND_LINE_SIZE);
/* 커맨드 라인 문자들을 복사한다 */
if(argc >= 2) {
    p = params->u.cmdline.cmdline;

    for(i = 1; i < argc; i++) {
        strcpy(p, argv[i], COMMAND_LINE_SIZE);
        p += strlen(p);
        *p++ = ' '; /* 스페이스문자로 분리한다 */
    }

    p--;
    *p = '\0'; /* 커맨드라인 문자열의 끝을 표시한다 */
}
/* 넘겨온 커맨드라인이 있다면 tag에 등록한다. */
if(strlen(params->u.cmdline.cmdline) > 0) {
    params->hdr.tag = ATAG_CMDLINE;
    params->hdr.size = (sizeof(struct tag_header) +
                       strlen(params->u.cmdline.cmdline) + 1 + 4) >> 2;

    params = tag_next(params); /* 다음 tag 구조체를 가리킨다 */
}
}

```

A.4. setup_initrd_tag() 함수

setup_initrd_tag() 함수에서는 Initial RAM Disk(압축된 Ram Disk)에 대한 tag를 설정한다. 우선 initrd tag라는 것을 알려주기 위해 tag필드에 ATAG_INITRD를 주고, initrd의 start(시작주소)에 RAMDISK_RAM_BASE 를, initrd 크기에는 INITRD_LEN으로 설정한다. 마지막으로 params포인터가 다음 tag 구조체를 가리키도록 만든다.

\$ (BLOBTOP)/src/blob/linux.c 소스중

```

static void setup_initrd_tag(void)
{
    params->hdr.tag = ATAG_INITRD;
    params->hdr.size = tag_size(tag_initrd);

    params->u.initrd.start = RAMDISK_RAM_BASE;
    params->u.initrd.size = RAMDISK_FLASH_LEN;

    params = tag_next(params);
}

```

A.5. setup_ramdisk_tag() 함수

setup_ramdisk_tag() 함수는 위에서 설정한 initrd(압축된 램디스크)가 압축이 풀렸을 때의 정보 tag를 설정하는 함수이다. 시작 주소에는 0을, 크기로는

RAMDISK_SIZE를 주고, flags에 1을 주어 자동으로 RAM disk 를 로드하도록 한다.
 역시, 마지막작업으로 params포인터가 다음 tag 구조체를 가리키도록 만든다.

\$(BLOBTOP)/src/blob/linux.c 소스중

```
static void setup_ramdisk_tag(void)
{
    /* an ATAG_RAMDISK node tells the kernel how large the
     * decompressed ramdisk will become.
     */
    params->hdr.tag = ATAG_RAMDISK;
    params->hdr.size = tag_size(tag_ramdisk);

    params->u.ramdisk.start = 0;
    params->u.ramdisk.size = RAMDISK_SIZE;
    params->u.ramdisk.flags = 1; /* automatically load ramdisk */

    params = tag_next(params);
}
```

A.6. setup_end_tag()함수

ATAG_NONE을 을 넣고 크기에 0을 주어 Tag의 마지막을 표시하기 위한 함수로 tag 설정함수 setup_XXX_tag함수중 가장 마지막으로 불리워 져야한다.

\$(BLOBTOP)/src/blob/linux.c 소스중

```
static void setup_end_tag(void)
{
    params->hdr.tag = ATAG_NONE;
    params->hdr.size = 0;
}
```

B. exit_subsystems(void) 함수 호출

그 다음 exit_subsystems()함수 호출로 __exitlist 영역에 등록된 콜백함수들을 실행한다.

__initlist 매크로와 같이 __exitlist매크로를 사용하여 __exitlist_start에서 __exitlist_end사이에 설정된 함수 포인터를 사용하여 모든 함수를 실행시킨다.

src/blob/initcalls.c에서 __exitlist 매크로를 사용하여 disable_icache()를 등록하고 있으며, disable_icache()함수에 대한 소스코드는 lib/icache.c에 존재한다. 이 disable_icache()함수는 코프로세서에 접근하여 ICache를 disable시키는 인라인 어셈블리어로 구성되어 있다.

\$(BLOBTOP)/src/lib/init.c 소스중

```
static void call_funcs(initlist_t *start, initlist_t *end,
                      u32 magic, int level)
```

```

{
.....
}

.....
void exit_subsystems(void)
{
    int i;

    /* call all subsystem exit functions */
    for(i = INIT_LEVEL_MAX; i >= INIT_LEVEL_MIN; i--)
        call_funcs((initlist_t *)&__exitlist_start,
                    (initlist_t *)&__exitlist_end,
                    EXIT_MAGIC, i);
}


```

4. BLOB 컴파일하기

위에서 BLOB 소스를 따라가며 수정해야 하는 곳을 알아보았다. 여기서는 이런 수정들이 다 끝나고 BLOB을 컴파일하여 BLOB 이미지를 생성하는 방법을 알아보도록 하자. 이런 과정을 모두 마치고 BLOB이미지가 생성되면 이 이미지를 부트로姆에 JTAG등(사실 ROM 라이터기보다는 JTAG가 더 많이 사용될 것이다.)을 이용하여 써넣으면 된다.

4.1. make clean명령

make clean명령은 make명령으로 만들어진 화일들을 지운다.

// 우선 위에서 작업된 이란 환경설정된 것들을 지우기 위하여 make distclean 명령을 내//린다. 이 명령은 환경설정되었던 것을 지워주는 역할을 한다.

4.2. configure 명령

처음 우리의 보드이름을 추가하는 작업을 했다. 이를 환경화일에 추가하기 위하여 configure화일을 다시 만들어야 하며, 이 작업을 수행하는 것이 rebuild-gcc 명령이다. 따라서, 우리는 이 명령을 먼저 수행한 후 configure 명령을 실행해야 한다.

4.2.1. rebuild-gcc 명령수행

이제 \$(BLOBTOP)/configure.in로 부터 \$(BLOBTOP)/configure화일을 만들어야 하는데 이를 확인하기 위하여 configure화일을 지우자. 1장의 설치에서 보았듯이 configure.in화일로 부터 configure실행화일을 만들기 위해서 \$(BLOBTOP)/tools/rebuild-gcc를 실행하였었다. 실행해 보자.

```

[root@Dci blob-xscale]# ./rebuild-gcc
$Id: rebuild-gcc,v 1.1.1.1 2001/06/27 19:47:42 erikm Exp $
Setting up for use with GNU C/C++ compilers

```

```
[root@Dci blob-xscale]# ls configure
configure
```

4.2.2. configure 명령

만약, 위의 과정이 제대로 되어 있지 않다면, 유효하지 않은 보드이름이라는 에러 메시지가 뜰 것이다.

우선 크로스 컴파일러를 등록하고 그 아래를 수행하며 제대로 되었을 경우 나타나는 메시지로 아래와 같이 나온다면 성공이다.

☞ Bash 의 경우

```
[root@Developers blob-2.0.5-pre2]# export CC=/usr/local/arm/arm-linux-gcc
[root@Developers blob-2.0.5-pre2]# export OBJCOPY=/usr/local/arm/arm-linux-objcopy
```

☞ tcsh 의 경우

```
- setenv CC /usr/local/arm/arm-linux-gcc
- setenv OBJCOPY /usr/local/arm/arm-linux-objcopy
```

```
[root@Developers blob-2.0.5-pre2]# ./configure --with-board=kelb --with-cpu=pxa255 --with-
linux-prefix=/usr/src/kernel/linux-2.4.19 arm-unknown-linux-gnu
Configuration
```

```
-----
Target cpu                pxa255
Target board              Korean Embedded Linux Board
C compiler                /usr/local/arm/bin/arm-linux-gcc
C flags                   -Os -I/usr/src/kernel/linux-2.4.19/include -Wall -march=armv4 -
mtune=strongarm1100 -fomit-frame-pointer -fno-builtin -mapcs-32 -DCPU_pxa255
Linker flags              -static -nostdlib
Objcopy tool              /usr/local/arm/bin/arm-linux-objcopy
Objcopy flags             -O binary -R .note -R .comment -R .bss -S
Ethernet support          none
USB support               no
Clock scaling support     no
Memory test support       no
Debugging commands support no
LCD support               no
MD5 support               no
Run-time debug information no
```

혹시 위의 명령으로 host를 못찾는 다고 에러가 나고, 위와 같은 메시지를 볼 수 없다면, configure 옵션에 --host=i386-linux-gnu를 추가해보도록 하자.

4.3. make 명령으로 컴파일 하여 보자.

에러 메시지없이 \$(BLOBTOP)/src/blob디렉토리에 blob화일이 생성되었는가?

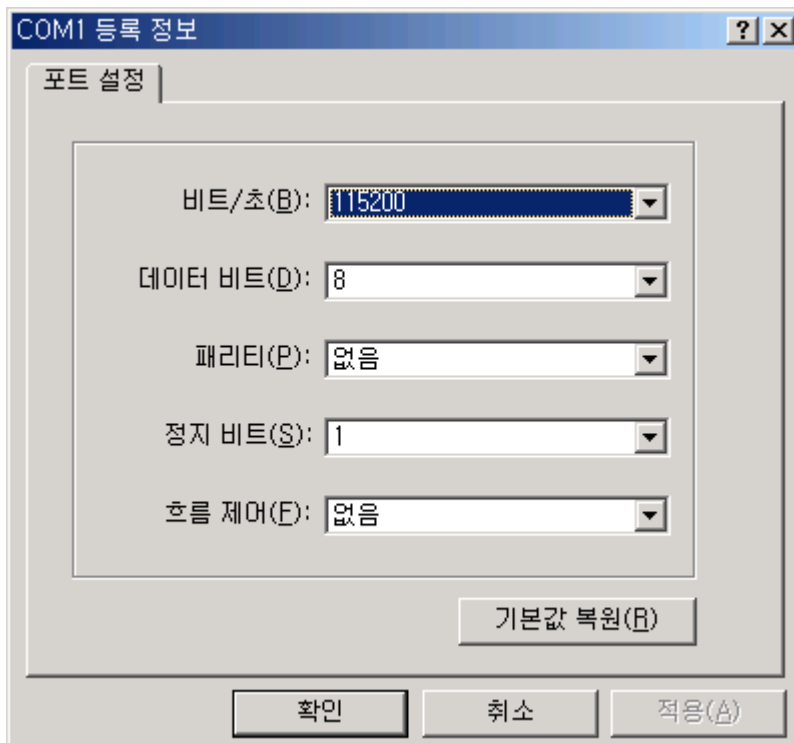
5. BLOB 사용하기

생성되었다면, JTAG를 이용해서 이를 플래시의 0x00000000 번지 영역에 써 넣는다.

이제 PC의 COM1포트와 보드의 FFUART에 시리얼 케이블을 연결하도록 한 후 시리얼통신을 할 수 있는 Window의 하이퍼터미널이나 리눅스의 Minicom을 실행하도록 한다.

5.1. 시리얼 설정

아래는 Window시스템하의 하이퍼터미널을 이용한 설정화면이다. 부트로더 수정중 보레이트를 115200으로 주었었으므로 아래와 같이 설정한다.



5.2. 부팅메시지 확인

자 이제 전원을 켜보자. 아래와 같이 메시지가 나오는가? 아래는 부팅도중 키를 눌러 메뉴모드로 들어온 모습입니다. 나오지 않는다면, 천천히 소스의 수정부분을 살펴해보도록 한다.

```

wince - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)

Consider yourself LARted!

blob version 2.0.5-pre2 for Korean Embeded Linux Board
Copyright (C) 1999 2000 2001 Jan-Derk Bakker and Erik Mouw
blob comes with ABSOLUTELY NO WARRANTY; read the GNU GPL for details.
This is free software, and you are welcome to redistribute it
under certain conditions; read the GNU GPL for details. Modified by Jang Seon Wg
Memory map:
  0x04000000 @ 0xa0000000 (64 MB)
Loading blob from flash . done
Loading kernel from flash .... done
Loading ramdisk from flash ..... done
Autoboot in progress, press any key to stop .....
Autoboot aborted
Type "help" to get a list of commands
blob> _

```

5.3. blob 메뉴설명과 파일다운로드 방법

help를 치면 아래와 같이 메뉴항목이 나온다.

```

wince - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)

Autoboot in progress, press any key to stop .....,
Autoboot aborted
Type "help" to get a list of commands
blob> help
The following commands are supported:
* reset
* reboot
* boot
* download
* xdownload
* flash
* speed
* status
* reload
* reblob
* fwrite
* ferase
* tftp
* ifconfig
* help
Use "help command" to get help on a specific command
blob>

```

메뉴의 모든항목의 도움말은 “help reset”과 같이 주어 볼 수 있다.

① Reset

터미널 화면을 지워준다.

② reboot, boot 메뉴

시스템보드를 부트시킨다.

③ Download

시리얼 포트로 **Uuencode**를 사용하여 **blob**, **kernel**, **ramdisk**를 다운로드할 수 있다. 따라서, 시리얼 포트로 위와 같이 볼 수 있다면, **JTAG**를 사용하지 않고 시리얼을 통하여 **blob**, **kernel**, **ramdisk**를 다운로드 할 수 있다. 다음 **xdownload**와 방식이 비슷하니 아래를 참조하기 바란다.

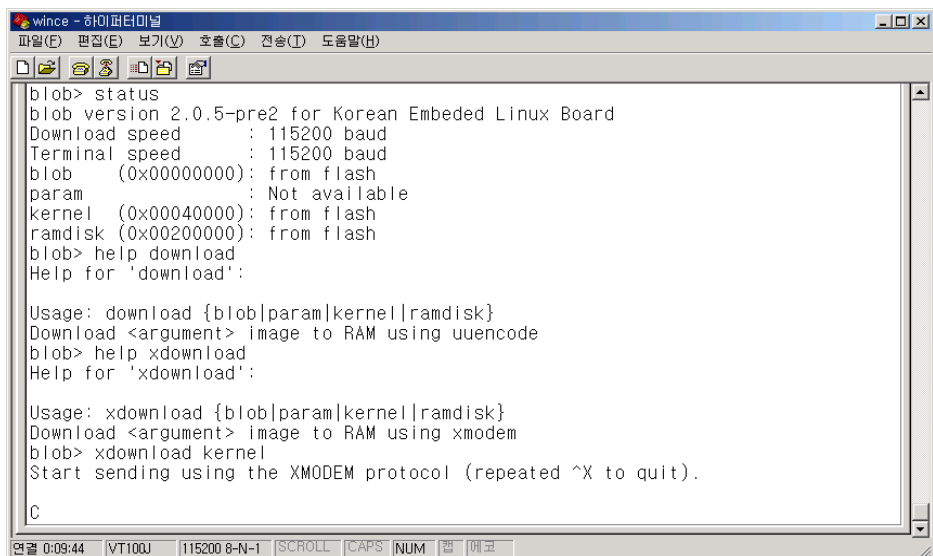
④ Xdownload

■ 사용법 : **xdownload [blob | kernel | ramdisk]**

Xmodem을 사용하여 **blob**, **kernel**, **ramdisk**를 다운로드 받는다.

예를 들어 커널이미지인 **zImage**를 플래쉬의 설정영역에 넣기 위한 방법은 아래와 같다.

- 시리얼로 다운로드 받기 위해 아래와 같이 “**xdownload kernel**”라고 명령을 준다. **Blob**의 경우는 “**xdownload blob**”, 램디스크의 경우는 “**xdownload ramdisk**”로 주면된다.



```

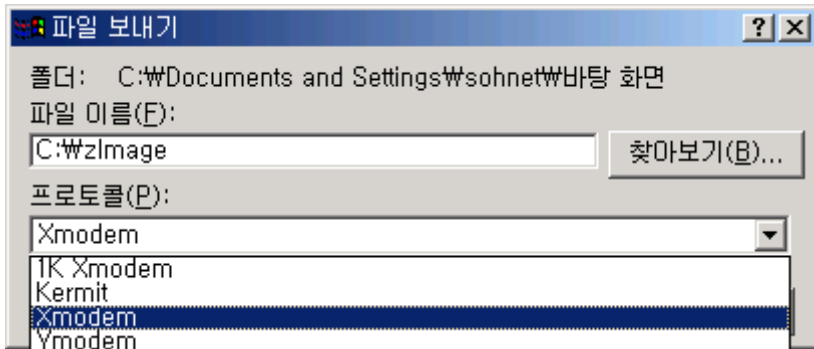
wince - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)
[Icons]
blob> status
blob version 2.0.5-pre2 for Korean Embedded Linux Board
Download speed      : 115200 baud
Terminal speed     : 115200 baud
blob (0x00000000): from flash
param              : Not available
kernel (0x00040000): from flash
ramdisk (0x00200000): from flash
blob> help download
Help for 'download':

Usage: download {blob|param|kernel|ramdisk}
Download <argument> image to RAM using uuencode
blob> help xdownload
Help for 'xdownload':

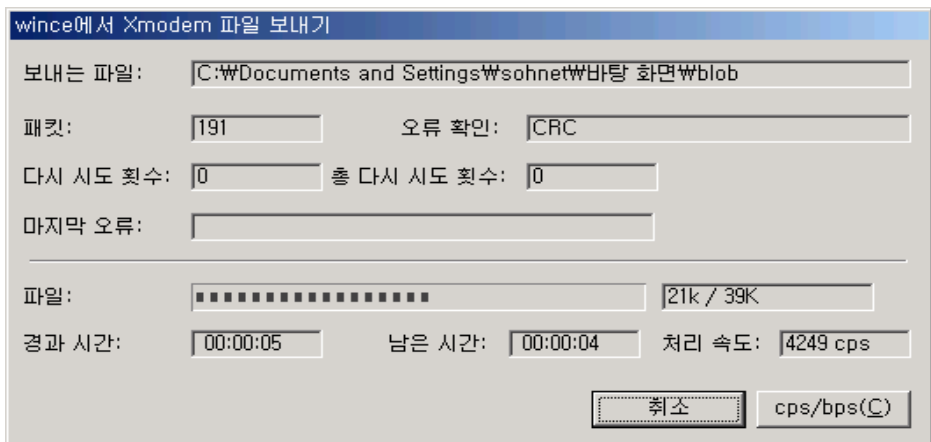
Usage: xdownload {blob|param|kernel|ramdisk}
Download <argument> image to RAM using xmodem
blob> xdownload kernel
Start sending using the XMODEM protocol (repeated ^X to quit).
C
연결 0:09:44 VT100J 115200 8-N-1 SCROLL CAPS NUM 캡 메코

```

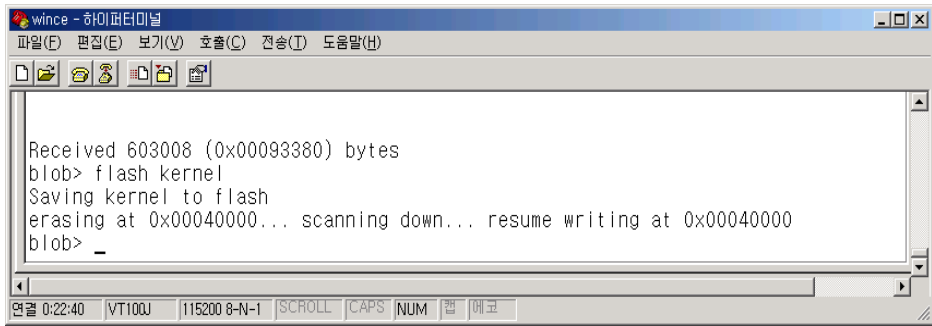
- 하이퍼터미널의 전송 메뉴에서 파일보내기를 선택한 후 찾아보기에서 전송할 이미지인 **zImage**를 선택하고 프로토콜로 **Xmodem**을 선택한다.



- 보내기 버튼을 눌러 전송을 시작하면 아래와 같이 전송이 이루어 질 것이다.



- 지금까지의 작업으로 램의 커널영역에 커널이 다운로드 되었다. 이를 플래쉬롬에 써 넣기 위해서는 다음에 보게될 “ flash “ 명령을 사용한다. 커널이미지를 플래쉬에 쓸 것이므로 아래와 같이 “flash kernel”로 명령을 준다. 플래쉬에 이미지를 쓰는데 에러가 생긴다면 부트로더 소스의 플래쉬에 대한부분이 자신의 보드와 맞는지 확인해야 한다. 다음은 에러가 없이 쓰여진 결과이다. 보다시피 우리가 정해 커널영역으로 정해 놓은 0x40000번지에 쓰고 있다.



⑤ speed, status, reload, reblob

이 명령들은 보레이트 수정(speed), 상태정보표시(status), 플래쉬에서 램 영역으로의 다시 로드하기 위한 명령(reload), blob의 재시작(reblob)등을 나타낸다.

⑥ fwrite, ferase

■ 사용법 : **fwrite** [소스주소] [타겟주소] [길이]

메모리의 특정영역에서 내용을 플래쉬에 써넣거나 플래쉬의 특정영역을 지우기 위하여 사용한다. 사용법은 아래에서 설명할 것이다. Flash명령과 다른 점은 flash 명령의 경우 소스주소와 타겟주소가 정해져있지만, fwrite의 경우에는 소스주소와 타겟주소를 임의로 줄 수 있다.

⑦ tftp, ifconfig

이 명령들은 “configure”명령시 “--with-eth= ”와 옵션을 사용하여 랜카드를 살렸을 경우 나타난다. Blob-xscale은 Lan91c111을 기본적으로 지원하고 있으므로, 쉽게 살릴 수 있다. 아래와 같이 하면 된다. 파일의 수정에 대해서는 CD롬의 src/blob/smc91x.h와 /src/blob/smc91x.c파일을 참조하라.

```
[root@Developers blob-2.0.5-pre2]# ./configure --with-board=kellb --with-cpu=pxa255 --with-linux-prefix=/usr/src/kernel/linux-2.4.19 --with-eth=smc91x
[root@Developers blob-2.0.5-pre2]# make
```

랜카드를 살렸다면, ifconfig 명령으로 IP를 설정한다.

```

wince - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)

blob> ifconfig
Mac addr      : 08.00.3e.26.0a.5b
Our IP addr   : 192.168.0.45
Server IP addr : 192.168.0.10
blob> ifconfig ip 192.168.0.99
blob> ifconfig server 192.168.0.10
blob> ifconfig
Mac addr      : 08.00.3e.26.0a.5b
Our IP addr   : 192.168.0.99
Server IP addr : 192.168.0.10
blob> _

```

서버의 IP는 TFTP서버(리눅스 호스트 컴퓨터)의 IP를 설정해 준다.

TFTP서버가 제대로 설정되어 있다면 이제 **tftp**명령을 통하여 파일을 다운로드 받을 수 있다. TFTP서버의 설정에 대해서는 개발환경설치 편을 참조하도록 하자.

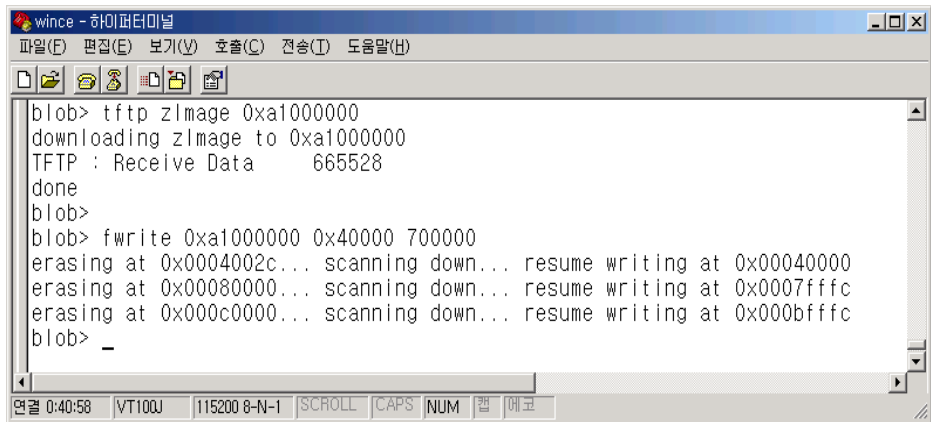
```

wince - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)

blob> tftp zImage 0xa1000000
downloading zImage to 0xa1000000
TFTP : Receive Data 665528
done
blob>
blob> _

```

위에서 0xa1000000번지의 램영역에 **zImage**파일을 다운로드 받았다. 이를 이제 **fwrite**를 사용하여 커널이미지의 해당영역(0x40000 ~ 0x140000)에 써넣도록 한다. 아래와 같이 한다.



```
wince - 하이퍼터미널
파일(F)  편집(E)  보기(V)  호출(C)  전송(T)  도움말(H)

blob> tftp zImage 0xa1000000
downloading zImage to 0xa1000000
TFTP : Receive Data 665528
done
blob>
blob> fwrite 0xa1000000 0x40000 700000
erasing at 0x0004002c... scanning down... resume writing at 0x00040000
erasing at 0x00080000... scanning down... resume writing at 0x0007ffff
erasing at 0x000c0000... scanning down... resume writing at 0x000bffff
blob> _
```

연결 0:40:58 VT100 115200 8-N-1 SCROLL CAPS NUM 캡 메코

6