Qplus-P Target Builder 사용자 매뉴얼

Qplus-P Target Builder User's Manual

Version 1.1

2002 년 8 월 2 일

강우철(wchkang@etri.re.kr)

윤희철(hcyun@etri.re.kr)

ETRI
한국전자통신연구원

목 차

1. 이 문서에 대해

Qplus-P(QplusP) 는 현재의 리눅스에 부족한 실시간 스케줄링, 저전력 관리, 관련 디바이스 드라이버등을 보완하여 정보가전기기에 사용될 수 있도록 전자통신연구원(ETRI)에서 개발중인 확장된 리눅스 운영체제이다.

이 문서는 Qplus-P 를 사용하여 정보가전기기에 적제될 시스템 소프트웨어들을 개발하는 과정을 도와주는 Qplus-P 타겟 빌더의 동작방식에 대한 설명과 이를 이용한 리눅스 타겟 시스템개발 과정에 대해서 기술한다.

2. 타겟 빌더 소개

Qplus-P 타겟빌더는 GUI(Graphical User Interface) 방식으로 동작하는 임베디드 리눅스 개발 도구이며, 개발자가 설정을 필요로하는 커널 / 응용의 각종 사항들을 옵션화시키고 그 옵션들간의 의존성검사와 필요한 정보를 인터렉티브하게 보여줌으로써 올바른 설정을 할 수 있도록 돕고 최종적으로 타겟 이미지에 적재되어 수행될 타겟 이미지를 자동 생성하는 기능을 제공한다.

Qplus 타겟빌더는 다음과 같은 기본 기능을 제공한다.

- 커널과 응용 패키지의 설정가능 사항의 옵션화를 통한 미세설정의 지원
- 각 옵션간의 의존성 자동 검사
- 커널과 응용의 동시 설정을 통한 커널과 응용간의 의존성에 대한 자동검사
- 타겟에 탑제 될 파일들의 리스트를 설정 가능하게 하고 라이브러리의 사이즈를 최적화 시킴으로써 최종 탑제 이미지의 사이즈를 줄임
- 유저가 쉽게 설정을 하게 도와주는 GUI 방식의 유저 인터페이스
- 타겟에 탑제 될 루트파일시스템 이미지의 자동 생성
- 타겟 초기화 스크립트와 네트웍 주소, 유저정보와 같은 기본 설정사항의 자동 생성

타겟 빌더는 응용의 소스컴파일과 배포에 널리 사용되는 RPM 기술과 리눅스 커널 설

정 기술로 채택될(2.5x 버전에서 채용 예정) CML2 기술을 사용하여 응용패키지/커널의 추가/변경을 쉽게한다.

본 장에서는 타겟 빌더의 간단한 사용방법에 대해서 설명한다.

2.1. 설치

이 장에서는 타겟빌더의 설치 및 삭제과정에 대해 기술한다.

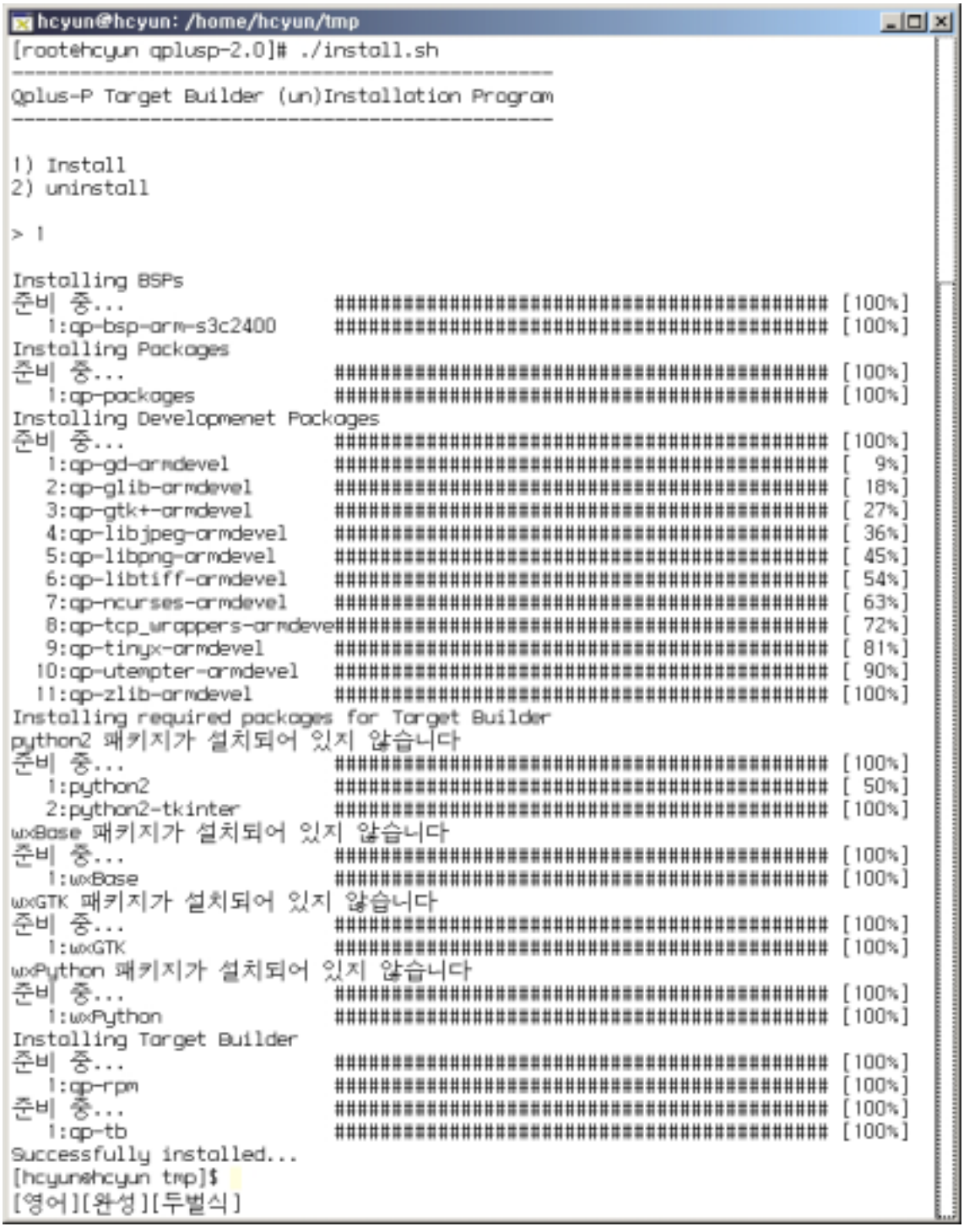
시스템 요구사항

타겟 빌더가 동작하기 위해서는 다음과 같은 요구사항을 만족시켜야 한다.

- 호스트 시스템의 배포판 : redhat 7.0 이상.
- 호스트 시스템의 성능: 가능한 고사양이 바람직.

설치

타겟 빌더의 설치를 위해서는 배포되는 qplusp-2.0.tar.gz 를 푼후에 qplusp-2.0 디렉토리로 이동한다. 다음으로 root 로 로긴한 후 install.sh 을 수행한다. 실행결과는 다음과 같다.



NOTICE: 만일 **python2.1** 버전이 설치되어 있었다면 이를 지운 후에 **install.sh** 을 수행하기 바란다. 타겟 빌더는 **python2.2** 버전이 필요로 하기 때문이다. **Python1.5** 만 설치되어 있는 경우에는 **install** 스크립트가 알아서 **python2.2** 를 설치하기 때문에 문제가 없다.

설치 후 디렉토리 구조

설치후 디렉토리 구조는 다음과 같다.

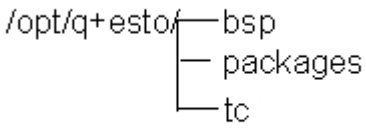


그림 1. 디렉토리 구조

bsp 디렉토리는 특정 아키텍쳐 및 보드를 지원하기 위한 커널, deploy 관련 툴 및 타겟 보드용으로 미리 컴파일된 패키지들의 이미지 파일이 있으며, packages 디렉토리는 Qplus-P 의 응용 프로그램들의 source rpm 과 QPD 파일이 있다. 마지막으로 tc 디렉토리에는 타겟 빌더 프로그램 및 관련 프로그램이 수록되어 있다.

2.2. 타겟 빌더의 실행

루트(root) 권한으로의 변경

타겟 빌더는 동작중 시스템 파일들의 접근등의 작업을 수행하므로 루트(root)권한으로 동작 시켜야 한다. root 계정으로 로그인 하거나 쉘 프로프트 상에서 다음과 같이 권한을 변경한다.

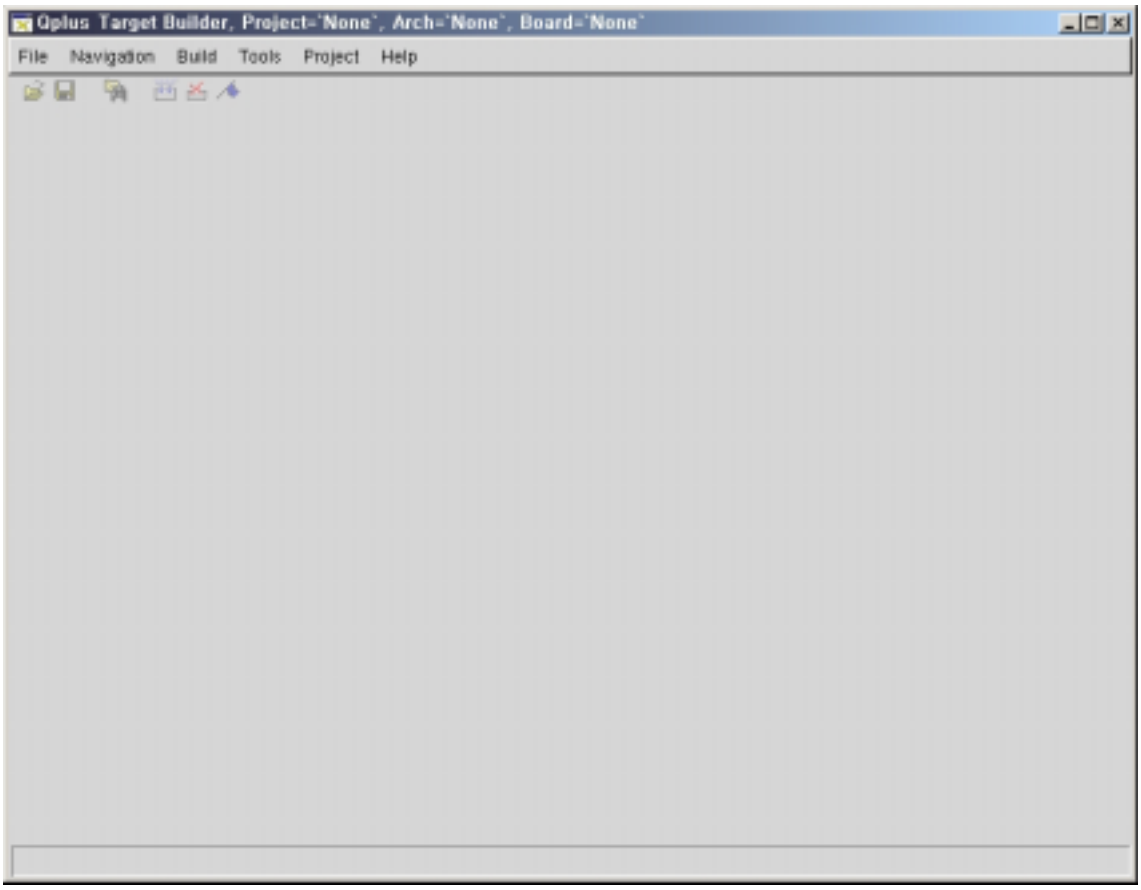
```
su -
```

타겟 빌더 (디폴트 설치 디렉토리는 /opt/q+esto/tc/bin)를 수행한다.

```
/opt/q+esto/tc/bin/tb
```

2.3. 프로젝트 생성

타겟 빌더를 처음 동작시키게 되면 다음과 같은 아무 내용도 갖지 않는 윈도우를 만나게 된다.



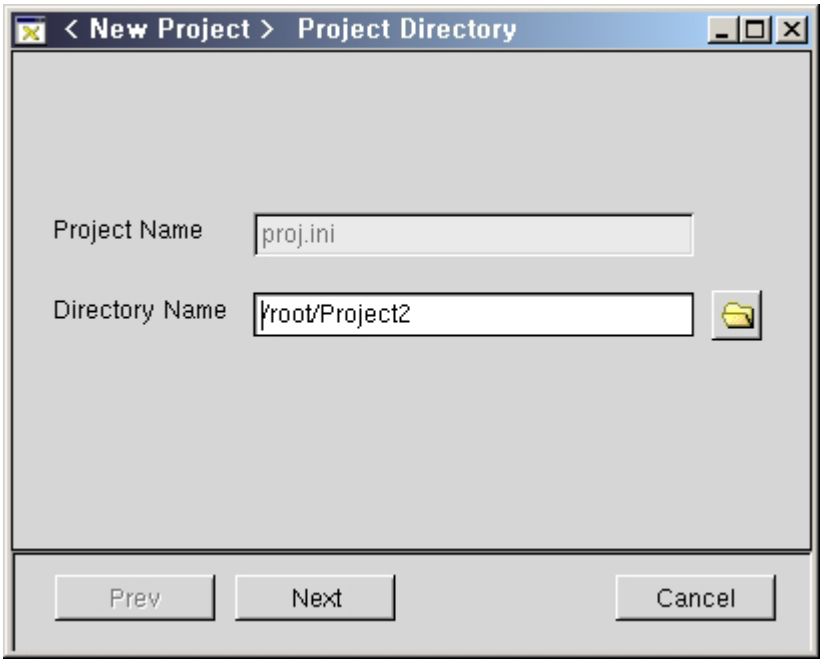
타겟 빌더는 프로젝트 단위로 관리되는데 개발자는 프로젝트를 생성해서 그 안에서 여러가지 설정과 빌드 작업을 하게 되며, 다른 설정을 갖는 환경에서 작업을 하고자 하는 경우 새로운 프로젝트를 생성해서 작업해야 하는 것이다. 위 그림의 상태는 아직 프로젝트가 하나도 생성되지 않은 경우이다.

새로운 프로젝트를 생성하기 위해서 프로젝트 이름과 디렉토리 타겟 아키텍쳐와 보드등을 지정해야 한다.

메뉴상에서 다음과 같은 순서로 선택한다.

```
Project > New
```

프로젝트 생성시 가장 먼저 다음과 같은 프로젝트 이름과 디렉토리 위치를 입력하는 창이 뜬다.



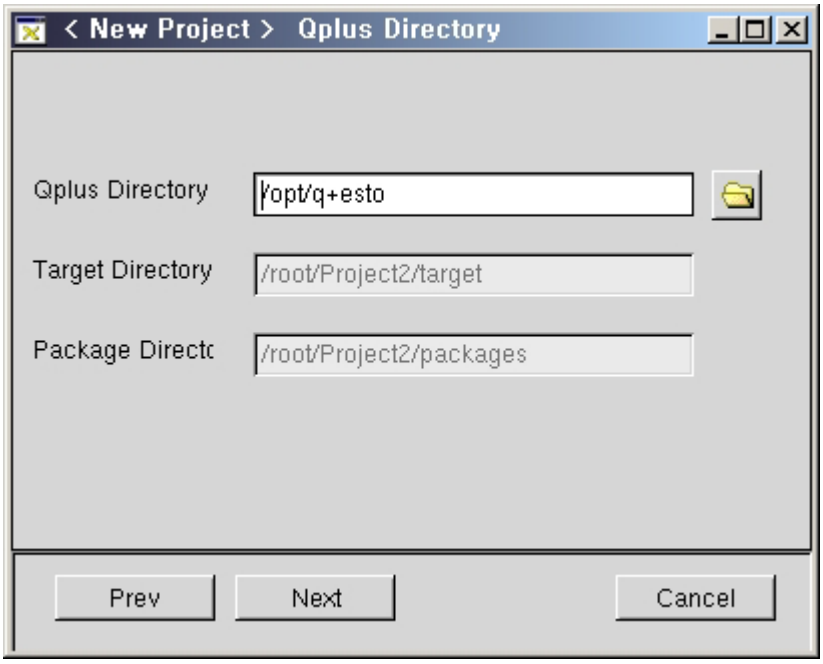
프로젝트를 생성할 디렉토리를 지정한다.

Project Name: 프로젝트 이름은 *proj.ini* 로 고정되어 있다.

Directory Name: 프로젝트를 만들고 싶은 디렉토리를 지정한다. 각 프로젝트는 서로 다른 디렉토리를 사용해야 한다. 이는 한 디렉토리에 여러 프로젝트를 생성할 수 없음을 말한다. 디폴트로 사용 가능한 프로젝트가 제시된다(홈 디렉토리 + ProjectXX).

프로젝트 디렉토리 지정이 끝났으면 **Next** 버튼을 누른다.

다음과 같이 디렉토리를 Q+ 디렉토리를 지정할 수 있는 창이 뜬다.



Qplus Directory 를 지정한다.

Qplus Directory: Qplus 타겟 빌더 관련 툴과 패키지가 설치되어 있는 디렉토리를 지정한다. 디폴트는 */opt/q+esto* 이며 **변경하지 않기를 권장한다**. 이 패쓰를 기준으로 여러가지 툴들이 하위 디렉토리에 설치되게 되며, 모두 이를 기준으로 동작한다.

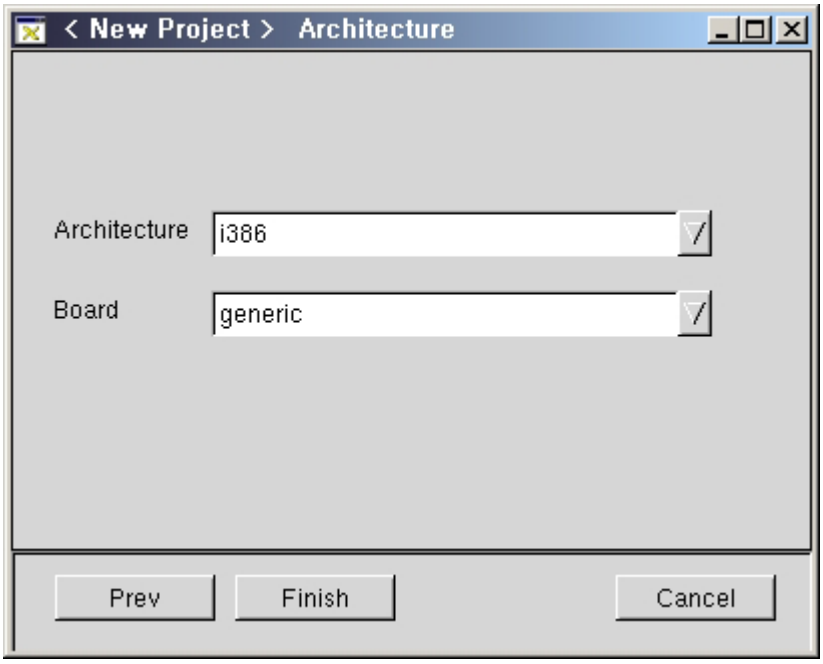
Target Directory: 프로젝트상에서 커널/응용을 빌드했을 경우 생성되는 커널 이미지와 타겟의 루트 파일 시스템은 임시적으로 이곳에 생성되며, 빌드후 이곳을 살펴 봄으로써 빌드 결과를 확인할 수 있다.

Package Directory: 각 패키지의 소스 RPM 파일들과 각 패키지의 옵션을 기술하는 QPD(Qplus Package Descriptor)[1]파일이 위치한다. 새로운 패키지를 추가하는 경우 이곳에 새로운 패키지의 소스 RPM 파일과 QPD 파일을 두어야 한다.

디렉토리 지정이 끝났으면 **Next** 버튼을 누른다.

[1] QPD 에 관한 내용은 5 장을 참조하라.

다음과 같이 타겟 아키텍쳐와 보드를 지정하는 창이 뜬다.



타겟 아키택쳐와 보드를 지정한다. 어떤 BSP(Board Support Package)가 설치되었는가에 따라 보여지는 값은 달라진다.

Architecture: 콤보 리스트에서 해당 타겟의 아키텍쳐를 지정한다. 현재 *arm* 과 *i386* 을 지정할 수 있다.

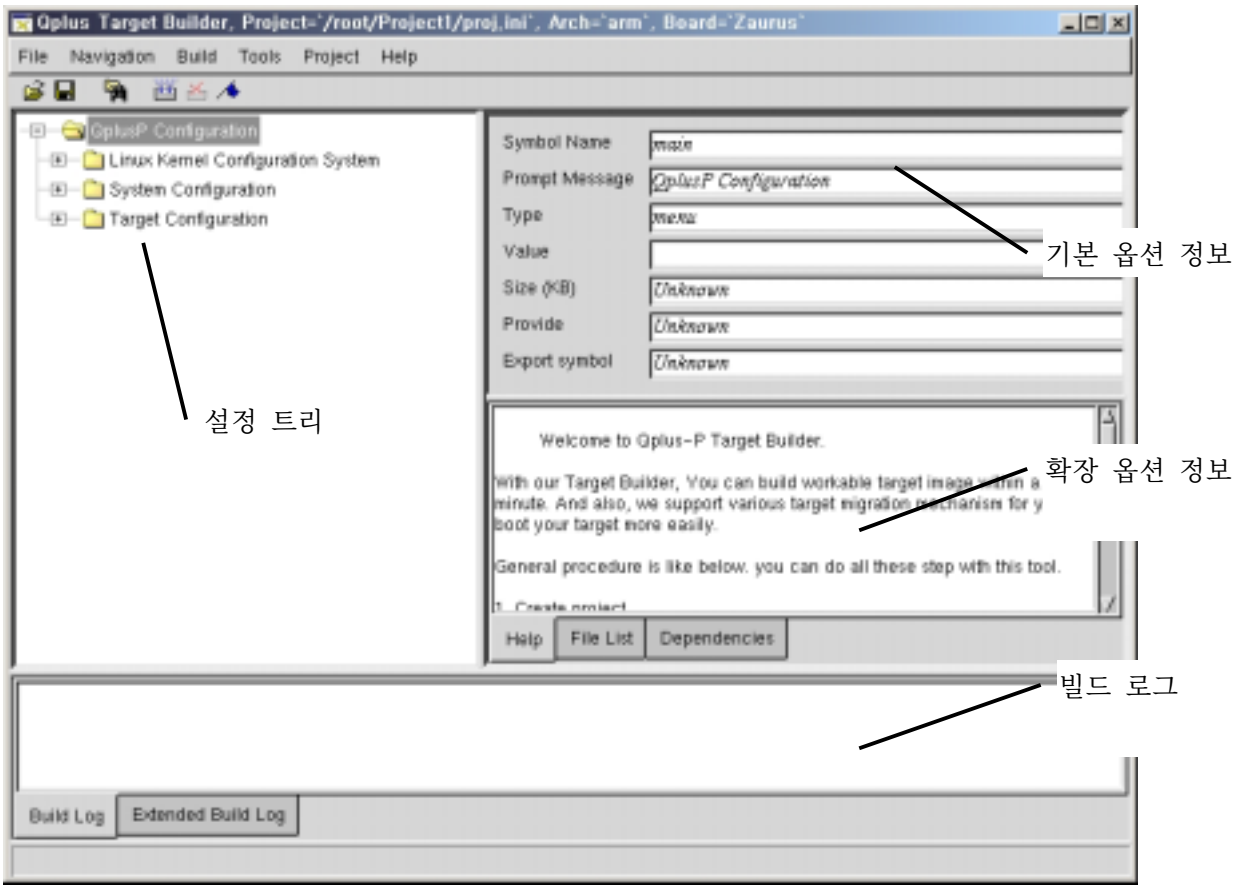
Board: 해당 아키텍쳐의 어떤 보드를 선택할지를 지정한다. 여기 지정된 값에 따라 BSP 가 프로젝트 디렉토리에 적절하게 설치되게 된다.

모든 설정이 끝났으면 **Finish** 버튼을 누른다.
이제 설정한 값에 따라 프로젝트를 생성하게 된다.

주의 : 프로젝트 생성은 시스템에 따라 **10~30 초 가량의 시간이 소요**된다. 이는 해당 설정에 따라 BSP 를 설치하고 옵션들이 가지는 규칙들을 새로 컴파일하기 때문이다.

2.4. 타겟 빌더 유저 인터페이스

다음은 프로젝트가 생성된 후 최초 화면이다.



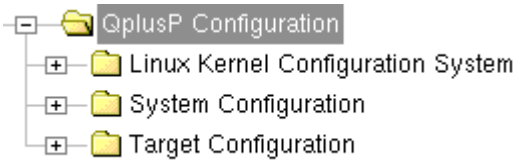
개발자는 설정 트리상에서 여러가지 아이템을 마우스로 클릭해볼 수 있으며, 이에 따라 관련된 각종 정보(도움말, 관련 의존성, 선택시 타겟에 설치될 파일 리스트)를 정보창을 통해서 확인해 볼 수 있다.

본 절에서는 유저 인터페이스 각 부분의 기능에 대해서 설명한다.

2.4.1. 설정 트리

커널, 응용 패키지, 타겟설치 방식등의 모든 설정 가능 사항을 트리 형태로 보여주는 창으로 이곳을 통해 개발자는 자신이 필요로 하는 아이템의 값을 설정하게 된다.

구조

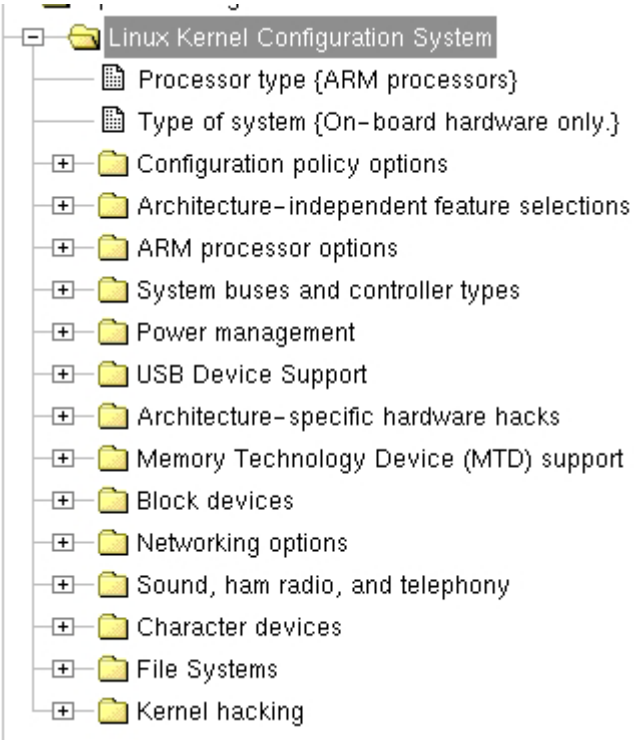


Qplus-P 는 옆의 그림에서 보는바와 같이 크게 리눅스 커널 설정, 시스템 설정, 타겟 설정의 3 가지 부분으로 나뉜다. 커널 설정은 일반적인 리눅스 커널의 설정 부분과 같으며 시스템 설정은 리눅스가 필요로 하는 각종 응용 프로그램들에 대한 설정이 이루어 지는 부분이다. 타겟 설정은 타겟 시스템이 정상적으로 수행되기 위해 필요한 각종

정보와 타겟에의 적재 방식등을 설정하는 부분이다.

각각의 설정 컴포넌트들에 대해서 다음에 설명한다.

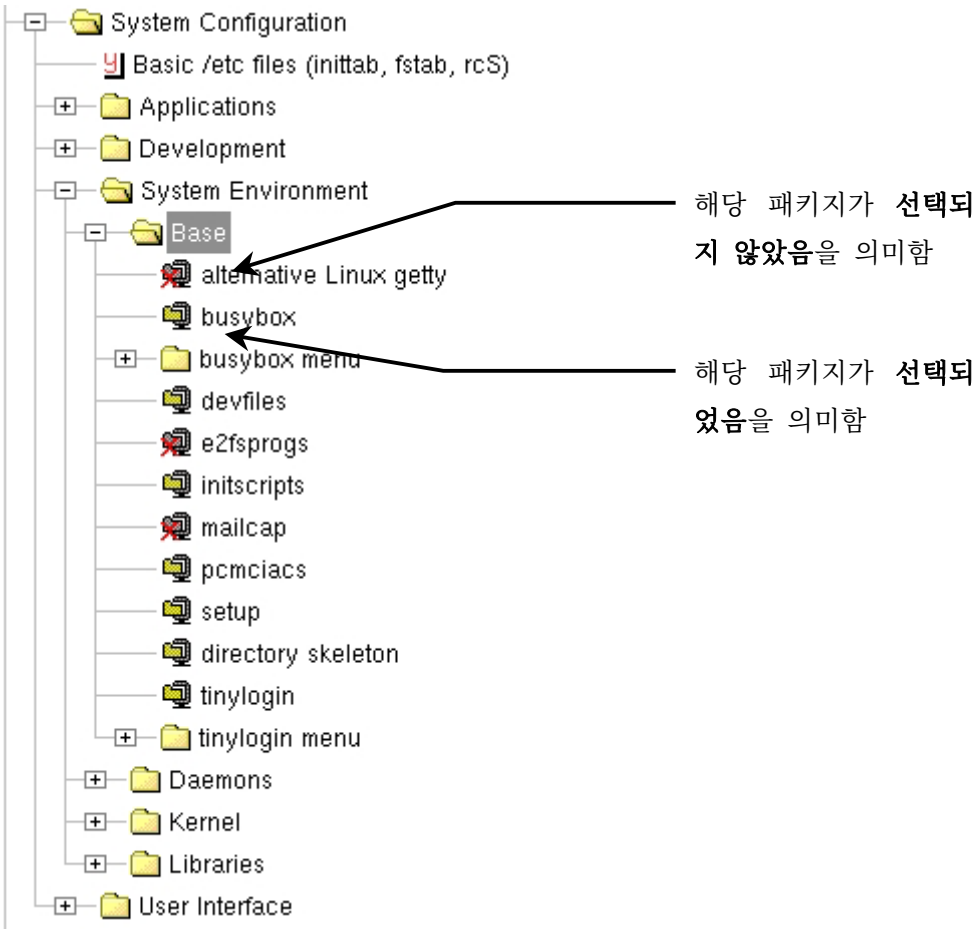
1. 리눅스 커널 설정부(Linux Kernel Configuration System)



리눅스 커널 설정에 관련된 부분으로 기본적으로 CML2 를 이용한 리눅스 커널의 설정 방식과 동일하다.

주의 : <Processor 타입> 메뉴는 프로젝트 생성시에 이미 결정될 것이므로 바뀌지 않음을 유의하기 바란다.

2. 시스템 설정부(System Cnfiguration)



시스템 설정부는 리눅스 시스템이 필요로 하는 각종 응용 패키지들의 설정을 할수 있는 부분이다. 각 패키지는 성질에 따라 각종 그룹(Applicatoins, Development, System Environment, Daemons, Libraries…)으로 분류되어 있다.

패키지는 설정 되었을 때와 그렇지 않을 때 다른 아이콘을 갖는다. 아이콘은 선택되었음을 의미하며 , 아이콘은 선택되지 않았음을 나타낸다.

각 패키지는 서브 메뉴를 가지는데, 한 예로 <*busybox menu*>는 *busybox* 메뉴의 서브 메뉴이다. 각 패키지의 서브 메뉴에서 관련 아이템들의 특성을 설정하면 타겟의 루트 파일시스템에는 여기에서 선택된 응용들만 컴파일되어 설치되게 된다.

주의 :

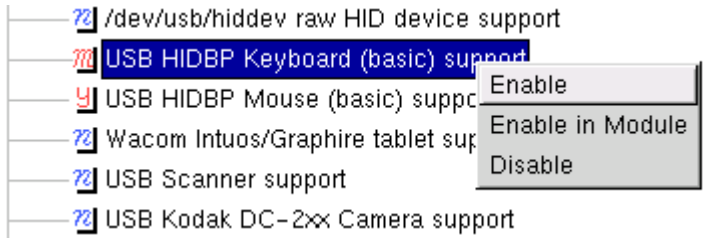
각 패키지의 서브 메뉴들은 패키지가 선택되어야 보이며 그렇지 않을 경우 트리상에 나타나지 않는다.

3. 타겟 설정(Target Configuration)

타겟 설정은 타겟 시스템이 필요로 하는 각종 정보와 타겟에의 적재(deployment)방식을 결정하는 부분이다.

각 아이템의 값 설정

각 설정 아이템들은 ***마우스의 오른쪽 버튼***을 누르면 아이템에 해당하는 메뉴가 뜨게된다.



BOOL 타입(Enable/Disable 만 결정할 수 있는 아이템)의 경우 ***"Enable"/"Disable"*** 메뉴만 뜨며, TRIT 타입(커널의 옵션중 커널 모듈로써 동작이 가능한 기능들)의 경우 ***"Enable in Module"***이 추가적으로 메뉴에 나타난다. 그 외의 타입(CHOICE, STRING, DECIMAL, HEXA)을 갖는 아이템의 경우 해당 값을 넣을 수 있는 대화 창이 뜨면 이를 통해 값을 입력할 수 있다.

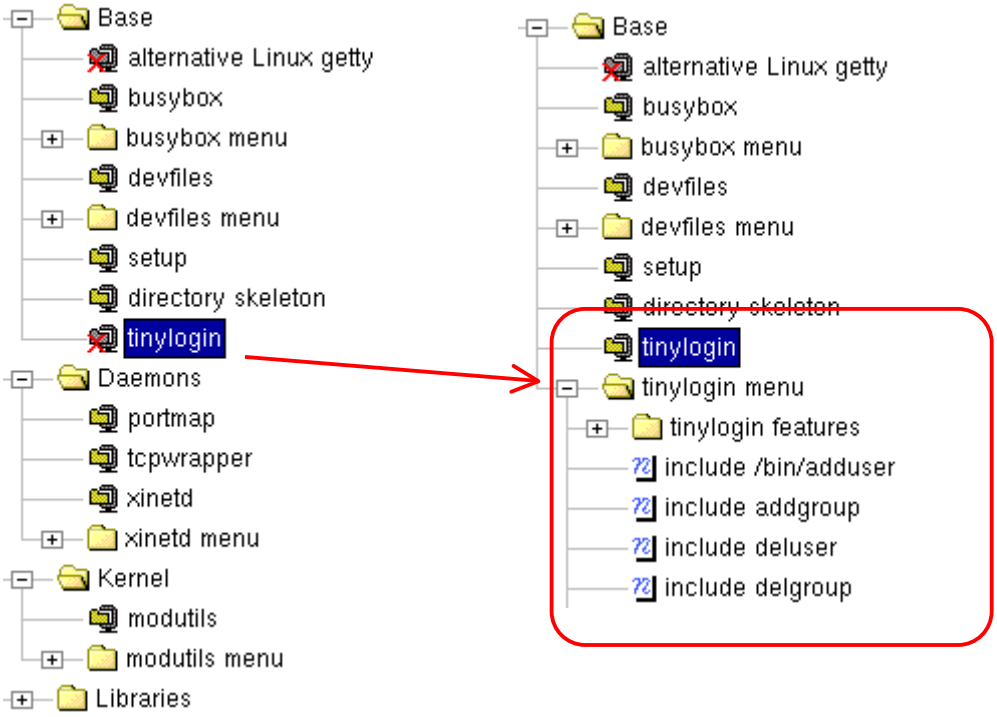
설정된 값은 아이콘으로 현재값을 다음과 같이 표시하게 된다.

- 아이콘 : BOOL, TRIT 타입인 아이템의 현재 상태가 **Enabled** 임
- 아이콘 : BOOL, TRIT 타입인 아이템의 현재 상태가 **Disabled** 임
- 아이콘 : TRIT 타입인 아이템의 현재 상태가 **Module** 임
- Broadcast address {129.254.180.255} : BOOL,TRIT 이외의 타입을 가지는 아이템은 위의 예와 같이 현재 값이 { } 안에 표시되어 나타난다.

아이템 보이기와 숨기기

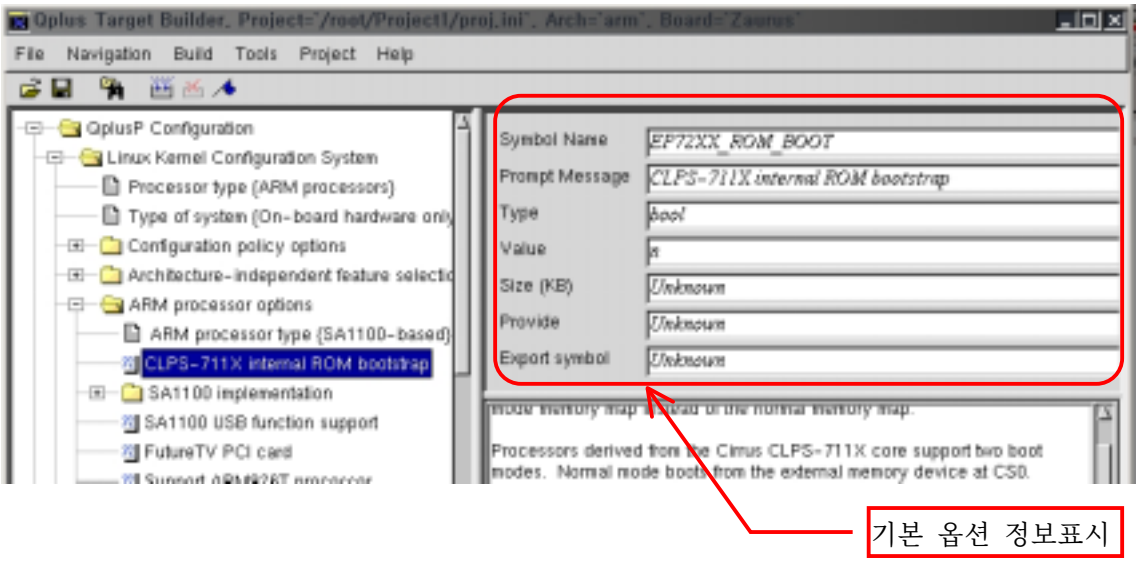
Qplus-P 타겟 빌더는 아이템을 점증적으로 보여주는 방식으로 동작한다. 점증적 보여주기란 현재 설정에 관련된 아이템들만 보여주고 그렇지 않은 부분은 감추는 것을 말한다. 이는 간단한 몇 개의 설정 아이템부터 시작해서 설정해 나가다 보면 관련된 아이템들을 점차적으로 보여주면서 복잡도를 늘려나가는 것이다. 리눅스 커널만 보더라도 현재 약 1750 개의 설

정 아이템을 가지고 있으므로 이를 한번에 보여주기 보다는 설정 방향에 따라 필요없는 부분은 감추고 꼭 설정을 필요로 하는 부분만 보여주는 것이 필요하다.
타겟빌더의 각 패키지는 이러한 보이기 규칙을 내재하고 있다(이는 QPD 파일을 통해 기술 가능하다).
한 예를 살펴 보자.



위의 그림에서 보는 바와 같이 *tinylogin* 패키지가 선택되지 않은 경우 *tinylogin* 이 가지는 설정 아이템들은 트리상에 보이지 않으나 *tinylogin* 을 선택한 경우 관련된 하위 메뉴들이 나타나게 된다.

2.4.2. 기본 옵션 정보



현재 선택된 아이템의 기본적인 정보를 표시한다. 표시되는 정보는 다음과 같다.

Symbol name: 타겟 빌더 내부적으로 각 아이템을 나타내기 위해서 사용되는 심벌의 이름이다. 사용자게는 큰 의미가 없다.

Prompt Message: 현재 설정 아이템에 대한 간단한 설명이다.

Type: 아이템의 타입값이다. 사용되는 타입은 *BOOL, TRIT, CHOICE, STRING, DECIMAL, HEXA* 이다.

Value: 아이템의 현재값이다.

Size: 패키지 아이템의 경우에만 의미가 있으며 해당 패키지의 foot-print 를 나타낸다.

Provide: 아이템간의 충돌을 감지하기 위해서 사용되는 것으로, 두개 이상의 아이템이 같은 심벌을 provide 할 경우 한 아이템만 선택되게 된다. Provide 필드는 현재 아이템이 Provide 하는 심벌이 무엇인지 보여준다.

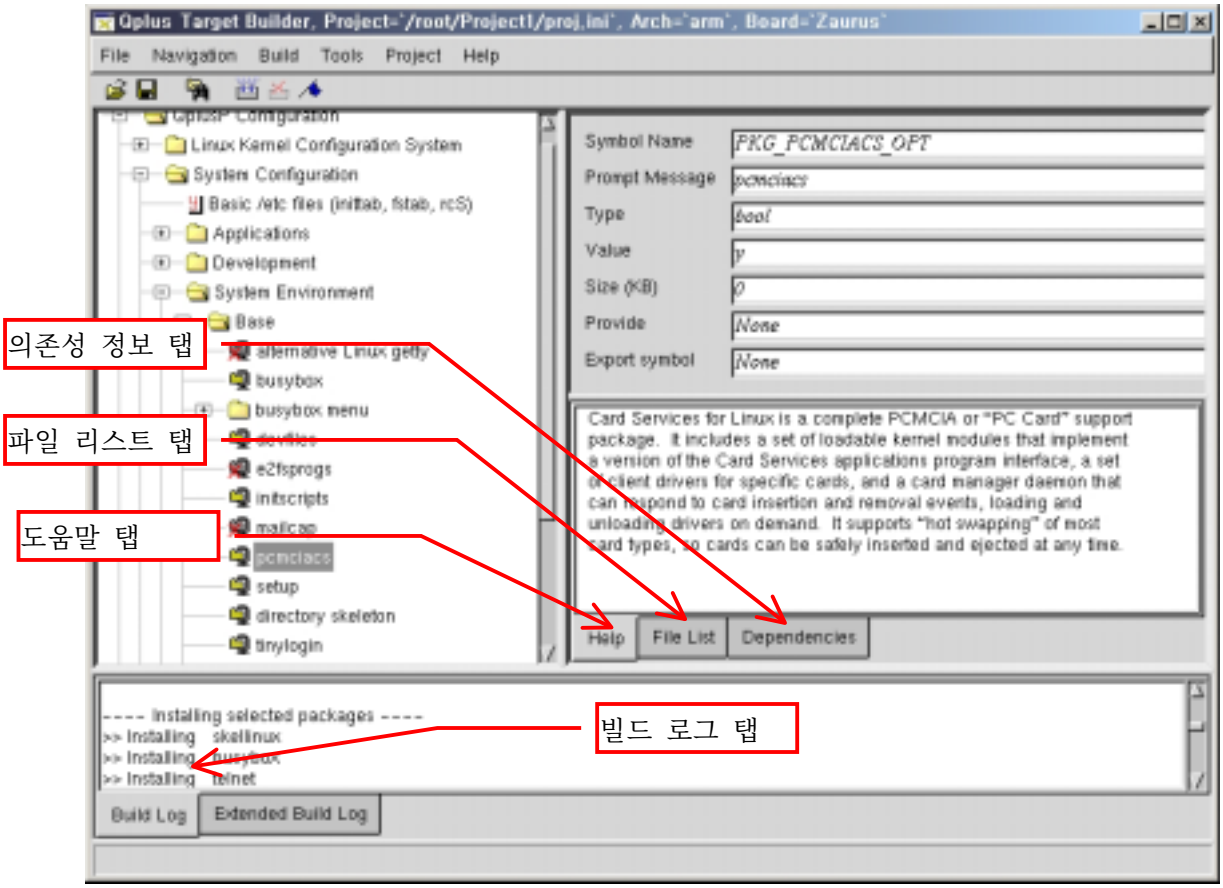
Export symbol: busybox, tinylogin 와 같은 일부 응용은 컴파일시에 특정 헤더파일에 기록된 심벌의 매크로 값에 따라 기능을 조절한다. 예를 들어 BB_WC 라는 심벌이 아래와 같이 정의된 경우 wc 명령을 busybox 에 포함시키게 된다.

```
#define BB_WC
```

타겟 빌더는 해당 아이템이 선택된 경우 헤더파일에 기록될 심벌 리스트를 가지고 있는데 export symbol 필드는 이 정보를 표시한다.

2.4.3. 확장 옵션 상태 정보와 빌드 로그 탭

설정트리에서 선택한 설정 아이템에 대한 자세한 설명, 파일 리스트, 관련된 의존성 정보를 보여주며 빌드 중에는 빌드 상태를 표시하는 탭이다.



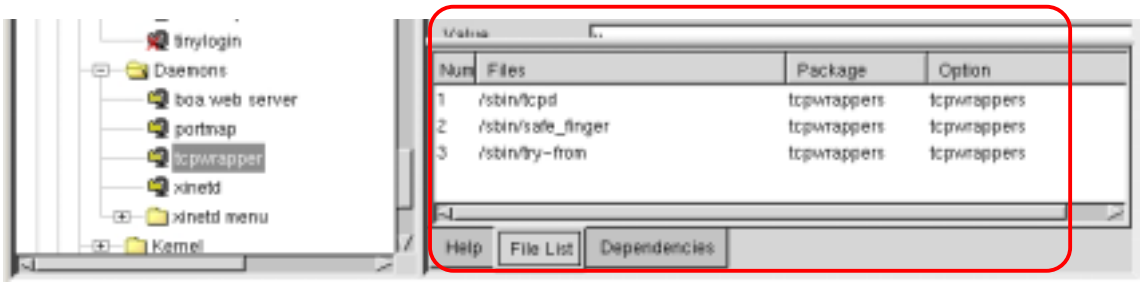
도움말 탭(help)

현재 설정된 아이템에 대한 자세한 정보를 얻을수 있다.

파일 리스트 탭(File List)

현재 선택된 아이템을 포함하는 하위 선택 아이템들이 타겟 루트 파일 시스템에 인스톨 될 때에 어떤 파일들이 들이 깔리게 될 지를 보여준다.

아래의 그림은 *tcpwrapper* 패키지가 선택되었고 이 패키지가 가지는 3 개의 파일 (*/sbin/tcpd, /sbin/safe_finger, /sbin/try-from*)이 타겟에 깔릴 것 임을 보여준다.



커널의 경우 하나의 커널 이미지이므로 각각의 설정 아이템이 파일리스트를 가지지 않는다. 최상위 아이템에 대한 파일리스트의 경우 타겟 시스템에 깔린 전체 파일리스트를 보여준다.

의존성 정보 탭(Dependencies)

각 아이템이 가지는 의존성 정보를 표시한다.
자세한 내용은 2.6 절을 참조하라.

빌드 로그 탭(Build Log)

빌드(커널, 응용, 타겟에의 적재)시에 빌드 상태 정보를 표시한다.

2.5. 설정값의 저장과 로드

2.5.1. 설정값의 저장

타겟 빌더의 각 프로젝트는 디폴트로 설정 상태를 프로젝트 디렉토리의 *config.out* 파일에 저장하며 프로젝트를 새로 오픈 할 때 이 파일에 저장된 설정 값을 읽어 오게 된다. 현재 설정값은 프로젝트의 종료, 빌드의 시작시에 자동으로 저장되며, 메뉴의 ***File*** > ***Save*** 를 통해 강제적으로 저장하게 할 수 있다.

2.5.2. 다른 파일로의 설정값 저장과 로드

타겟 빌더는 아이템들의 현재 설정 상태를 임의의 파일에 저장하고 로드하는 기능을 제공한다. 이는 다양한 설정을 해보고 이들 설정 상태를 저장한뒤 가장 적당한 설정을 나중에 다시 쓰고자 할 때 편리한 기능이다.

현재 설정의 저장(Save As)

메뉴의 ***File*** > ***Save As*** 를 이용해 현재의 설정을 저장할 파일을 선택한다.

현재 설정의 로드

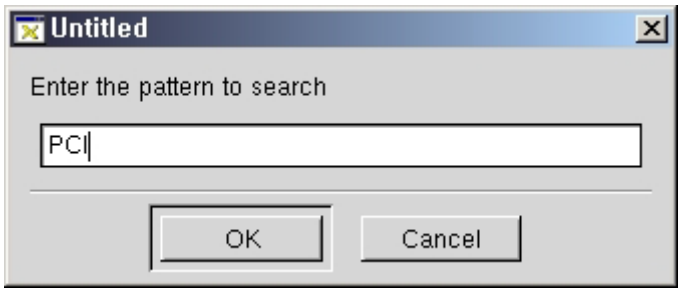
메뉴의 ***File*** > ***Load*** 를 이용해 설정 정보가 담긴 파일을 로드한다.

2.6. 찾기(Search) 기능

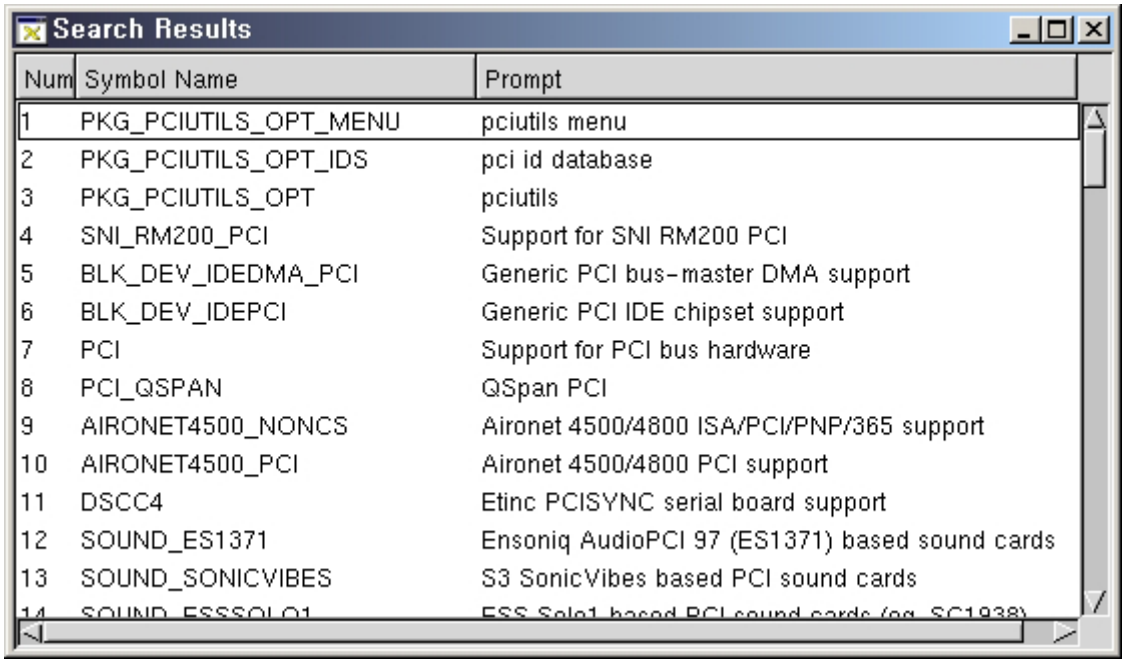
타겟 빌더는 도움말, 프롬프트 메시지, 내부 심벌이름중 특정 문자열을 가지는 아이템을 찾는 기능을 제공한다.

2.6.1. 메시지 찾기 기능(Search Symbols, 단축키 Alt+F8)

아이템중 프롬프트 메시지, 심벌이름에 특정 문자열을 가지고 있는 것을 찾는 기능이다.
예를 들어 PCI 라는 문자열을 가지는 아이템들을 모두 찾고 싶은 경우 메뉴의
Navigation > ***Search symbols*** 를 수행하면 다음과 같은 메시지 창이 뜬다.



위와 같이 메시지창에 원하는 문자열을 입력하고 OK 버튼을 누른다.



결과로써 위와 같이 PCI 라는 문자열을 심볼이름과 프롬프트 메시지에 가지는 모든 아이템들의 리스트를 보여주게 된다. 결과 값을 마우스로 클릭하면 트리상에서 포인트가 해당 아이템으로 바로 이동한다.

2.6.2. 도움말 찾기 기능(Search help)

Search Symbols 기능과 같으나 아이템의 도움말 파일을 검사해서 매칭되는 문자열을 가지는 경우 해당 아이템들의 리스트를 보여준다.

2.6.3. 바로가기 기능(Goto)

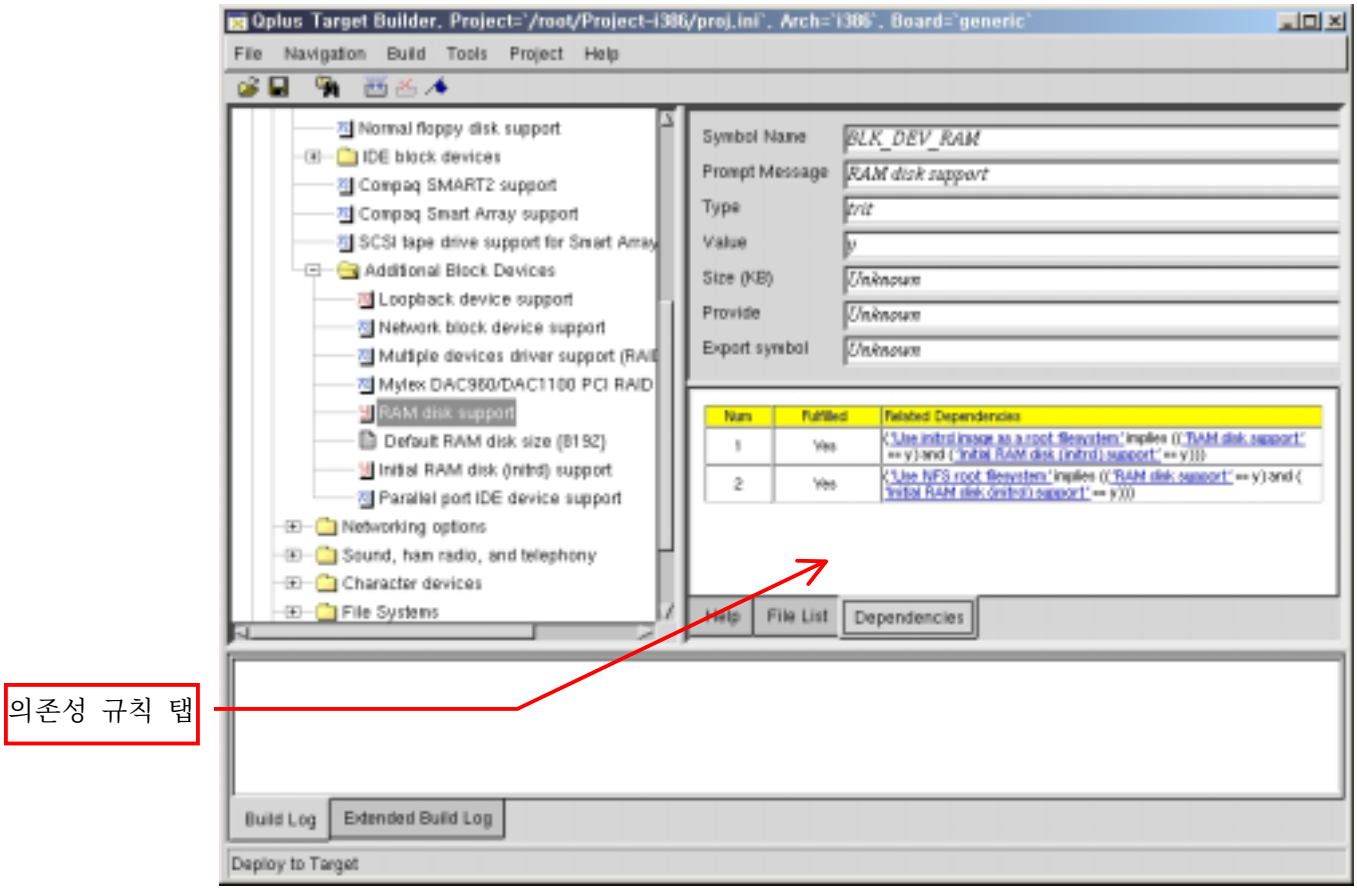
타겟빌더에서 아이템을 표현하기 위해 사용되는 심벌의 이름을 입력할 경우 트리상에 표시해 주는 기능이다.

2.7. 의존성의 표시

일부 설정 아이템들은 서로 의존성을 가지고 있다. 한 예로 *X windows* 와 *syslogd* 데몬은 내부 통신을 위해 커널의 *unix domain socket* 기능이 선택되어야만 제대로 동작한다. 이러한 설정 아이템 간의 의존성은 QPD[2](Qplus Package Descriptor) 파일내에 기술이 가능하다.

타겟 빌더는 각 설정 아이템에 연관된 모든 의존성을 ***Dependencies*** 탭을 통해 보여주고 관련된 설정 아이템으로의 바로가기 기능을 제공한다.

[2] QPD 파일의 작성에 대해서는 5 장을 참조하라



2.7.1. 의존성 규칙이 가지는 논리식

각 설정 아이템이 가지는 의존성 규칙은 올바른 설정에 대한 규칙을 논리식으로써 표현한 것이다.

논리식에 사용되는 기호는 다음과 같다.

and

or

not:

== , !=, > , < , >= , <=

Implies (포함 관계를 나타낸다. ⊃ 또는 →와 같은 의미임)

한 예로 Unix Domain socket 설정 아이템을 트리상에서 클릭하면 다음과 같은 2 개의 의존성 규칙을 가짐을 볼 수 있다.

.

('include syslogd' implies ('unix domain socket' == y))

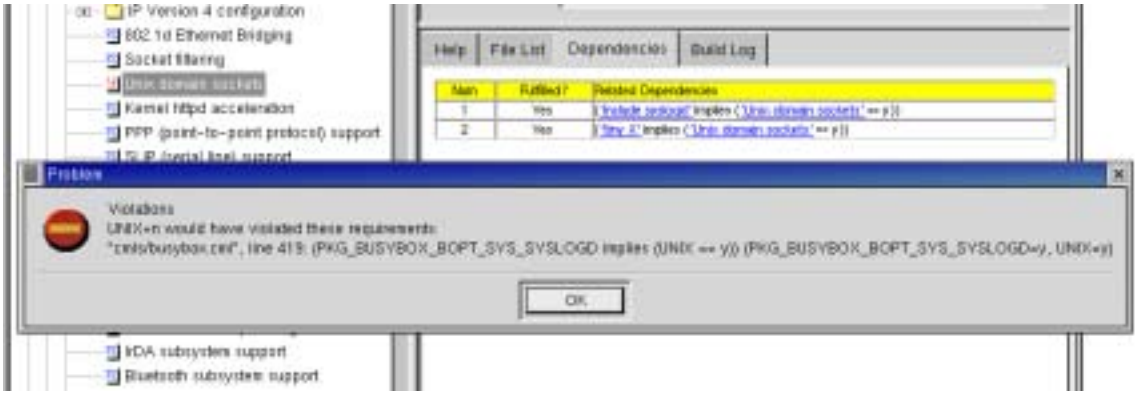
('tinyx' implies ('unix domain socket' == y))

이는 'syslogd' 아이템과 'tinyx' 가 선택되기 위해서는 'unix domain socket' 설정 아이템이 선택되어야 됨을 보여준다. 또한 관련된 각 설정 아이템들이 파란색으로 표시되어 있는 것은

하이퍼텍스트로써 클릭시 트리상에서 해당 아이템을 표시해 주게 된다.

2.7.2. 의존성 규칙의 위반

타겟빌더는 아이템의 설정 값이 변할때마다 관련된 의존성을 만족하는지 여부를 체크해서 위반이 발견되면 이를 알리고 변경된 값을 원래값으로 되돌린다.
위의 예에서 'tinyx'가 선택되면 자동으로 'unix domain socket'이 선택되게 된다. 그러나 이때 'unix domain socket'을 Disable 상태로 만들려고 하면 다음과 같은 경고 창이 뜨며 'tinyx'로 인해 'unix domain socket'이 필요하다는 것을 보여준다.



2.8. 빌드

타겟 빌더 내에서 직접 응용 패키지들과 커널을 컴파일하고 타겟에 적제될 루트파일시스템의 이미지를 만드는 기능이다.
Build 메뉴를 통해서 제공된다.

Build Kernel: 커널을 컴파일한다. 이전 상태값과 비교해 변경이 이루어진 경우만 재 컴파일하게 된다. Build > Build Kernel 메뉴를 사용하면 됨

Build Kernel –Force: 커널을 강제 컴파일 한다. 이는 커널의 변경값에 대한 체크가 제대로 이루어 지지 않을 때 사용하는 기능이다. Build > Build Kernel –Force 메뉴를 통해 사용하면 됨

Build Application: System configuration 내의 각 응용 패키지들을 컴파일 한다. 컴파일은 선택적으로 이루어지며 설정이 변경된 패키지중 새로 컴파일을 요하는 경우만 하게 된다. Build > Build Application 메뉴를 통해 사용 가능한다.

Build Root filesystem: 타겟에서 사용 가능한 루트 파일 시스템을 tar.gz 형식으로 생성해 준다. 디폴트로 프로젝트 디렉토리 밑의 target 디렉토리에 생성해 준다.

Build All: 커널 빌드, 응용 빌드, 루트 파일 시스템 생성의 3 가지 일을 차례로 수행해 준다.

Build Target deployment image: 타겟에 적재될 시스템 이미지를 생성해 준다. 자세한 설명은 4 장을 참조하라. Build > Build Target deployment image 메뉴를 이용한다.

Stop: 빌드 도중 정지시키는 기능이다. 메뉴의 Build > Stop 을 통해 수행된다.

주의 : stop 메뉴가 눌러져도 바로 빌드가 정지되는 것은 아니다. 이는 빌드가 불완전하게 끝날경우 다음의 빌드에 영향을 줄 수 있기 때문이다. **stop** 버튼을 누르고 현재의 빌드가 안전하게 종료될 때 까지 기다리기를 권장한다.

3. 타겟 이미지 생성

앞장을 통해 Target Builder 의 개략적인 사용방법을 알아 봤다. 이 장에서는 실제로 Target Builder 를 이용하여 여러분의 Qplus-P Linux 를 작성하여 본다. 따라서 이 장에서는 Qplus-P Linux 를 구성하는 Kenel 및 root 파일시스템의 작성과 관련된 설정 항목 및 이미지 작성 방법에 대해 다룬다.

3.1. 커널 및 응용 프로그램 설정

프로젝트를 생성하면 해당 BSP 의 기본 설정이 로딩된다. 기본 설정은 대부분의 패키지 및 기능을 활성화시킨 바로 Build 가능한 상태의 설정이다. 최초로 이미지를 생성하는 경우라면 기본 설정 그대로 사용하여 돌려보는 것이 바람직하다.

사용자는 이 기본 설정으로부터 Target Builder 의 GUI 를 이용하여 직접 설정을 변경할 수 있으며 또한 별도의 설정 파일을 로딩하여 적용할 수 있다. 개별 패키지 및 커널 설정은 온라인 도움말을 참조한다.

3.2. 네트워크 설정

타겟 시스템의 네트워크 관련 설정을 한다. 현재는 Static IP 설정만을 지원한다. 네트워크를 사용하지 않는다면 이 항목은 무시한다.

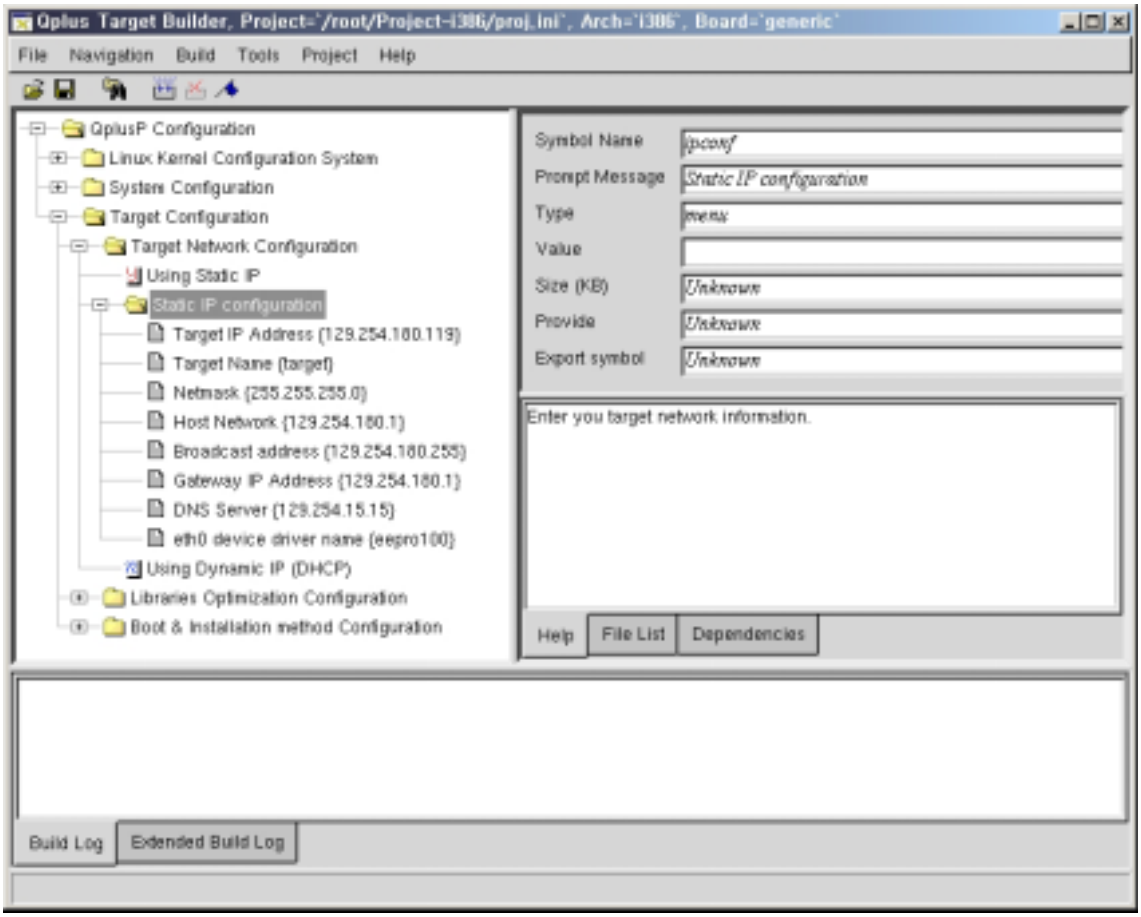


그림 2. 네트워크 설정

3.3. 라이브러리 최적화 설정

Target Builder 는 다음의 두 가지 라이브러리 최적화 방법을 제공한다.

File level optimization

Target root filesystem 생성시 모든 패키지의 실행 파일들을 조사하여 root filesystem 에 존재하는 실행 파일들이 동작하기 위해 필요한 공유 라이브러리 이외의 라이브러리 파일은 제거한다.

Symbol level Optimization

이 항목을 선택하면 Target root filesystem 에 존재하는 모든 실행 파일들이 사용하는 심볼들을 조사하여 사용되는 심볼들만을 포함하는 reduce 된 공유 라이브러리를 생성한다.

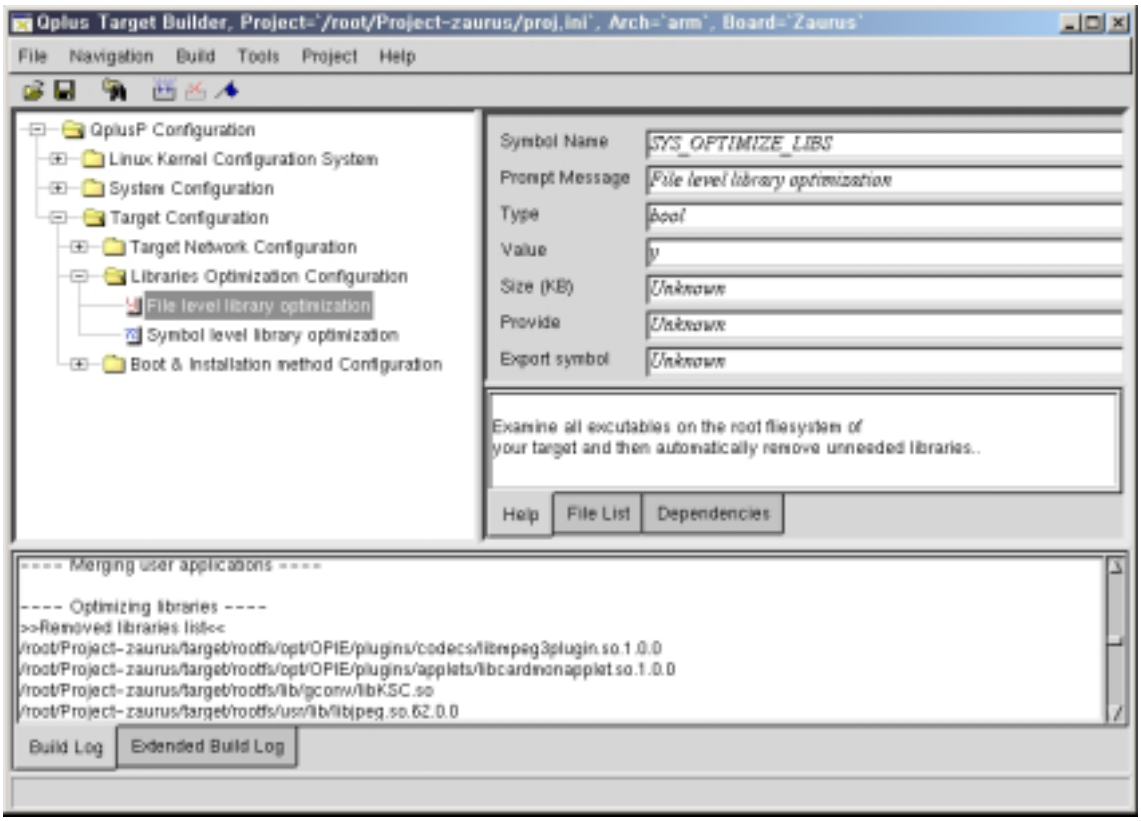


그림 3. 라이브러리 최적화 설정

일반적으로는 파일 수준의 최적화만으로도 충분하나 사이즈가 매우 중요한 경우 심볼 수준의 최적화를 수행하면 최소 크기의 이미지를 얻을 수 있다. 단 심볼 수준으로 최적화 된 경우 새로운 응용 프로그램의 추가를 위해서는 target builder 를 이용하여 전체 라이브러리를 다시 생성해 주어야만 한다는 단점이 있다.

사용자가 특정 라이브러리 파일을 optimizer 에 의해 삭제되지 않기를 원한다면 <projdir>/piclib/keeplist 파일에 다음과 같이 해당 라이브러리 파일을 이름을 적어둔다

```
lib/libnss_files-2.2.3.so
lib/libnss_dns-2.2.3.so
```

keeplist 파일의 예. GLIBC 의 name service 중 FILE 및 DNS 서비스 이용

위의 예에서 보여지는 glibc 의 nss 관련 library 파일들은 응용 프로그램에서 직접적으로 사용하는 심볼이 없기 때문에 library optimizer 에 의해 필요 없는 것으로 간주되지만 DNS 및 /etc/hosts 등의 정보를 이용하기 위해서는 반드시 필요하다. (기본으로 설정되어 있음)

3.4. 이미지 생성 (Build)

Deploy 설정을 제외한 모든 설정이 완료되었으면 이제 커널과 root 파일시스템을 생성할 수 있다.

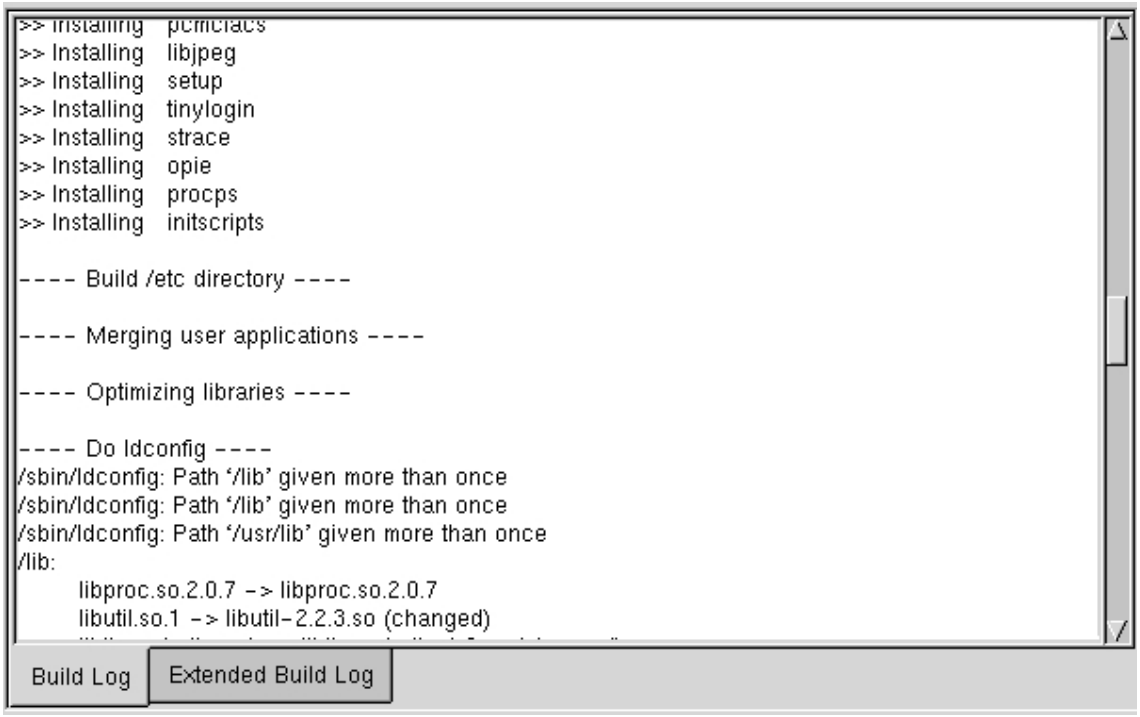
3.4.1. Build All

메뉴상에서 다음의 순서로 선택하거나 단축키 F8 을 누른다.

Build > Build All

Build All 은 그 아래의 Build Kernel, Build Application, Build Root Filesystem 을 모두 호출하여 커널 및 루트 파일시스템을 생성한다. 각각의 항목은 아래에 설명되어 있으므로 참고한다. 커널 및 응용 프로그램 build 시 이전 설정과 비교한 후 컴파일하기 때문에 컴파일 하는 경우는 거의 없으며 대부분의 경우 Build All 을 하여도 시간이 오래 걸리지 않는다.

Build 의 log 는 GUI 의 하단부의 Build Log 탭에서 확인할 수 있다.



다음은 Build 메뉴의 기타 항목에 대한 설명이다. 개별 응용 프로그램별로 컴파일 하는 것도 가능한데 이것은 응용프로그램을 선택한 후 마우스 오른쪽 버튼을 눌러 Build 를 선택하면 된다. 이경우 설정 여부의 변경과 상관없이 컴파일을 수행한다

Build Kernel

커널을 컴파일하여 <projdir>/target/kernel 디렉토리에 qplus 라는 이름으로 커널을 생성한다. 커널 컴파일 시에 이전의 커널 설정과 비교하여 변동사항이 없으면 컴파일을 하지 않는다.

Build Kernel – Force

설정의 변경 여부와 관계없이 무조건 커널을 컴파일 한다.

Build Application

선택한 응용 프로그램들을 컴파일 한다. Build Kernel 에서와 마찬가지로 이전 설정과 비교한 후 컴파일을 수행한다. 설정의 변경유무를 가릴 때 컴파일에 관계된 설정만을 가지고 비교를 하기 때문에 단순히 포함시킬 파일의 목록변경으로는 컴파일을 수행하지 않는다.

Build root filesystem

<projdir>/target/rootfs 디렉토리에 타겟 장치의 root filesystem 을 생성한다. Root filesystem 의 생성은 다음의 단계를 거친다.

- 앞서 컴파일 된 응용프로그램 설치
- 사용자의 설정에 따른 기본 설정 파일들 (/etc/init.d/rcS 등) 생성
- 라이브러리 최적화
- 타겟의 root filesystem 을 tar 파일로 묶어 <projdir>/target/rootfs.tar.gz 생성

생성된 rootfs.tar.gz 파일은 타겟의 root 장치에 설치하면 바로 사용 가능한 형태이다.

4. Deploy to Target (i386/Generic)

Deploy 란 호스트에서 타겟 빌더를 통해 만든 커널 및 루트 파일시스템 이미지를 타겟 하드웨어에 적재하는 것을 말한다. 이러한 Deploy 방법은 타겟 보드의 종류나 구성에 따라 매우 달라진다.

이 장에서는 일반적인 PC 환경을 가정하는 i386-generic BSP 를 사용했을 때의 deploy 방법에 대해 다룬다.

4.1. Host Requirement

홈서버와 Linux (Redhat 7.0 or 7.1 권장) 가 설치된 PC 가 필요하며 이들은 이더넷으로 연결되어 있어야 합니다. Linux PC (추후 '호스트' 라고 함) 에는 타겟 빌더를 설치하며 타겟 빌더를 이용하여 홈서버에 장착될 Qplus-P 시스템을 설정 및 생성하게 됩니다.

Linux PC 에서는 다음의 항목을 체크합니다.

1) loopback device 지원
   확인 방법은 root shell 상에서 다음이 성공적으로 수행되어야 합니다.

   ```
   # dd if=/dev/zero of=diskimage count=1024
   # mkfs.ext2 diskimage
   # mkdir mntptr
   # mount –o loop x mntptr
   ```

   마지막의 mount 가 성공적으로 수행되면 loopback 장치가 지원되는 것입니다.

2) minix filesystem 의 지원
   확인방법은 다음과 같습니다.

   ```
   # modprobe minix
   # cat /proc/filesystem
   ```

   목록중에 minix 가 있으면 됩니다.

3) dhcpd 설치 여부

/usr/sbin/dhcpd 파일이 있는지 확인

4) tftpd 설치 여부
/usr/sbin/in.tftpd 파일이 있는지 확인

5) nfs 서버 지원 여부
/etc/init.d/nfs start 가 수행되는지 확인

4.2. Etherboot Boot Floppy 디스크 만들기

Etherboot 란 단어 그대로 ethernet 을 통해 원격으로 부팅하는 것을 지원하는 패키지입니다. 기본적으로 홈서버는 Etherboot 클라이언트, 호스트 PC 는 부트 서버가 됩니다.

<projdir>/tools/etherbootimgs/ 디렉토리에서 확장자가 lzdsk 인 파일 중 자신의 타겟의 network card 와 일치하는 것을 골라 다음과 같은 순서로 디스크를 만든다.

1) 플로피 디스크를 삽입한다.
2) dd if=<device name>.lzdsk of=/dev/fd0
3) 만들어진 디스크를 타겟의 플로피 디스크에 집어넣고 부팅한다.

올바른 이미지를 이용하여 디스크를 만들었다면 Etherboot 는 랜카드를 찾은 후 bootp request 를 한다. 호스트쪽의 네트워크 설정은 4.4 장을 참조한다.

부팅시 화면에 랜카드의 MAC 주소가 나타나는데 이것을 기록해 두십시오. 다음에 나올 dhcpd 설정에 필요합니다.

4.3. Etherboot Boot CD-ROM 만들기

<projdir>/tools/etherbootimgs/ 디렉토리에서 확장자가 iso 인 파일 중 자신의 타겟의 network card 와 일치하는 것을 골라 다음과 같은 순서로 디스크를 만든다.

1) <devicename>.iso 파일을 CD-Writer 로 부터블 CD-ROM 으로 만든다.
2) BIOS 에서 First Boot 방법을 CD-ROM 으로 설정한다.
3) 만들어진 디스크를 타겟의 플로피 디스크에 집어넣고 부팅한다.

올바른 이미지를 이용하여 CD-ROM 을 만들었다면 Etherboot 는 랜카드를 찾은 후 bootp request 를 한다. 호스트쪽의 네트워크 설정은 4.4 장을 참조한다.

4.4. Etherboot 를 위한 호스트 설정

Etherboot 가 찾을 수 있도록 호스트에 관련 네트워크 서버 설정을 해 주어야 합니다. 필요한 서버는 dhcpd, tftpd, nfsd 의 세가지 입니다.

4.4.1. dhcpd 설정

아래의 예와 같이 /etc/dhcp.conf 를 설정합니다. 여기에서 hardware ethernet 은 타겟의 MAC 주소를 기록하며. Fixed-address 에는 타겟이 부팅시 사용할 ip address 를 기록합니다. 파일을 수정한 후에는 dhcpd 데몬을 재시작 합니다.

```
subnet 129.254.180.0 netmask 255.255.255.0 {
host homeserver {
        hardware ethernet xx:xx:xx:xx:xx:xx;
        fixed-address xxx.xxx.xxx.xxx;
}
}
```

/etc/dhcpd.conf 파일의 예

4.4.2. tftp 데몬 설정

다음과 같이 /etc/xinetd.d/tftp 파일에서 disable 을 no 로 설정합니다. 파일을 수정한 후에 xinetd 데몬을 재시작 한다.

```
service tftp
{
        disable = no
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot
              .
              .
}
```

/etc/xinetd.d/tftp 파일의 예

4.4.3. nfs 서버 설정

nfs 를 통해 export 할 디렉토리와 이를 마운트할 타겟의 주소를 설정해 주어야 합니다. 다음의 예는 /tftpboot 디렉토리의 모든 하위 디렉토리를 129.254.xxx.xxx 라는 타겟에 export 하는 예입니다. 역시 설정을 수정한 후에는 nfs 서비스를 재시작 합니다.

```
/tftpboot    129.254.xxx.xxx(rw,no_root_squash)
```

/etc/exports 파일의 예

4.5. Deploy with initrd root

앞서 네트워크를 통한 부팅을 위한 모든 준비를 마쳤다. 이 장부터는 에서는 실제로 타겟에 적재하는 다양한 방법들에 대해서 살펴본다.

Initial root 이 방법은 루트파일 시스템 이미지를 ramdisk 형태로 만든다. Ramdisk 의 최대 크기는 커널 설정에 따라 4M ~ 8M 로 제약이 있으므로 시스템의 크기가 작은 경우나 타겟 하드웨어의 테스트를 위한 목적으로 사용하는 것이 바람직하다. Initrd 방식을 사용하여 target 으로 deploy 하는 순서는 다음과 같다.

1. ‘Use Initrd as a root filesystem’ 옵션을 선택한다.

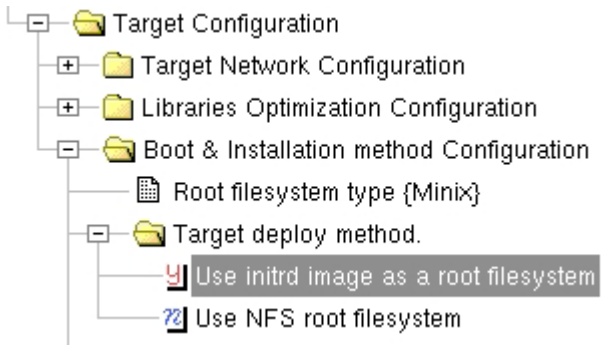


그림 4. Inird deployment option

2. 메뉴에서 Build > Build All 을 선택한다.
   앞서 커널 및 응용 설정에서 Build 를 수행하였다라고 Deploy 옵션에 따라 커널이나 응용 프로그램 설정이 자동으로 바뀔 수 있으므로 마지막으로 한번 더 Build All 을 해 주어야 한다.
   아래 그림은 initrd 를 선택하였을 때 자동으로 체크하는 dependancy 인데 initrd 방식을 선택하였을 때 커널을 ‘RAM disk support’ 와 ‘Initial RAM disk (initrd) support’ 옵션이 활성화 되는 것을 알 수 있다.

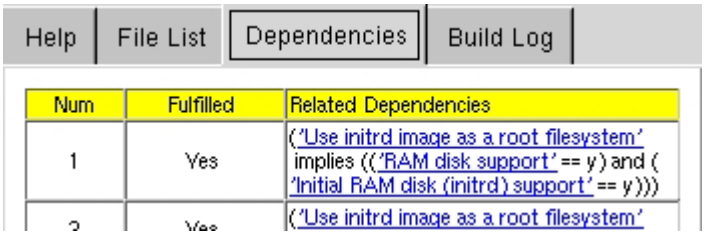


그림 5. Initrd 방식을 선택했을 때 자동으로 체크하는 dependancy

3. 메뉴에서 Build > Deploy Target Image 를 선택한다.
   에러없이 수행되면 /tftpboot 디렉토리에 다음과 같이 커널과 ramdisk 이미지가 포함된 etherboot 를 위한 이미지가 생성된다.
   /tftpboot/qplusp.etherboot
   커널과 ramdisk 각각은 <projdir>/target/kernel/qplus 과 <projdir>/target/rootfs.img 파일이다.

4. 호스트의 dhcpd 및 tftpd 설정을 확인한다.
   각 서비스의 설정은 4.4 장을 참조한다
5. 타겟을 부팅한다.
   이때 부트 플로피 또는 부트 CD-ROM 을 타겟에 삽입한 상태여야 한다. (4.2, 4.3장 참조)
   Etherboot 가 동작하여 Qplusp.Etherboot 파일을 서버로부터 다운받은 후 통상의 리눅스 부팅 메시지가 모니터 화면에 나타난다. (만일 타겟에 VGA, Keyboard 등이 없다면 Serial Console 을 사용한다. Serial Console 을 사용하는 것은 옵션 트리상에 'Use serial console' 항목을 Enable 하면 된다.)

4.6. Deploy with NFS root

호스트의 특정 디렉토리를 네트워크를 통해 타겟의 root 로서 사용하는 방법이다. 개발 호스트 상에서 파일들을 마음대로 추가 및 수정이 용이하기 때문에 개발 단계에서 가장 사용하기 적당한 방법이다. 기본 deploy 방법이다. 사용법은 다음과 같다.

1. Use NFS root filesystem' 옵션을 선택하고 그 하위 옵션 (nfs server 설정)들을 설정한다.

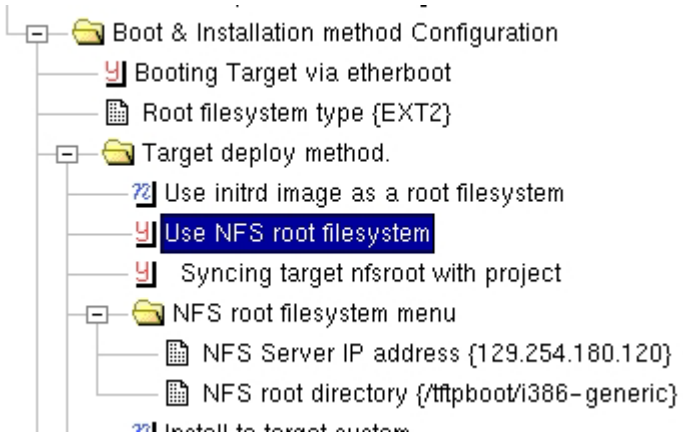


그림 6. nfs deployment option

'Use NFS root filesystem' 옵션을 선택하면 바로 아래에 'Syncing target nfsroot with project' 옵션과 그 아래 'NFS rootfilesystem menu' 메뉴가 나타난다.

'Sncing target nfsroot with project' 옵션을 enable 하면 rsync 를 이용하여 프로젝트 디렉토리에서 타겟 빌더에 의해 생성된 rootfilesystem 을 nfsroot 로 설정한 디렉토리와 (아래의 'NFS root directory' 옵션의 값) sync 를 한다. 이 옵션을 사용하지 않으면 생성된 root filesystem 을 tar archive 로 생성한다.

NFS root filesystem menu 에서 'NFS Server IP address' 는 nfs server (일반적으로 타겟 빌더를 수행하는 호스트와 동일한 호스트)의 ip address 를 입력하며, 'NFS root directory' 옵션은 타겟이 root 디렉토리로 사용할 호스트의 디렉토리 이름을 입력한다. 여기에서 설정한 정보는 타겟 부팅시 nfs server 에 대한 기본 설정 정보로 이용된다.

2. 메뉴에서 Build > Build All 을 수행한다.
3. 메뉴에서 Build > Deploy Target Image 를 수행한다.

Nfs deploy 에서는 /tftpboot/qplusp.etherboot 파일이 생성되고 위에서 'Syncing target nfsroot with project' 옵션을 설정한 경우 nfsroot directory 를 update 한다. Sync 를 설정하지 않으면 /tftpboot/rootfs.tar.gz 파일이 생성되는데 이것을 nfsroot directory 에 풀어주면 된다.

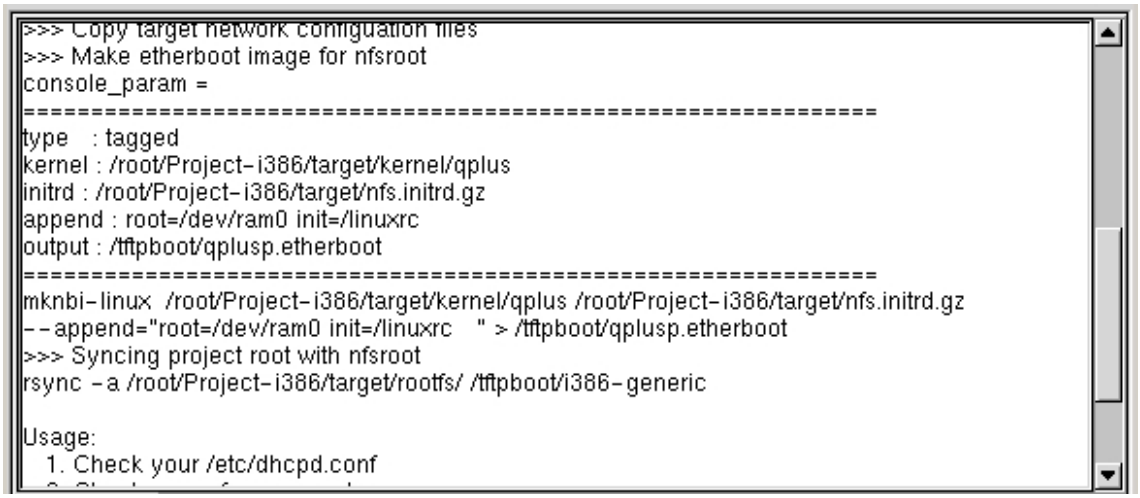


그림 7. nfs deploy 시의 log 화면

4. NFS 서버 설정을 확인한다.
   /etc/exports 에 호스트에 설치한 타겟의 root 파일시스템 디렉토리가 export 되었는지 확인한다. 자세한 것은 4.4.3장을 참고한다.
5. 타겟을 부팅한다.
   타겟을 부팅하면 화면 (모니터 또는 serial)에 다음과 같은 화면이 나타난다.

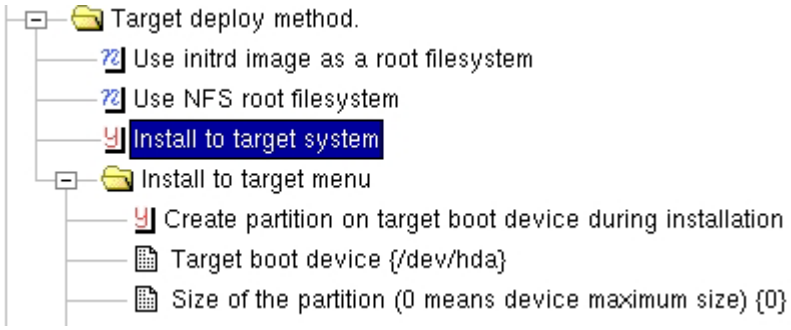
```
1) Using following default configuration
serverip    = <default serverip>
nfsrootdir = <default nfsrootdir>
2) Manual configuration
x) Exit to shell
>> 1
```

앞서 타겟빌더상에서 한 nfs 설정 정보가 올바르다면 1 을 입력하여 기본 설정을 사용하면 된다.

4.7. Install to target hard disk

이 방법은 호스트에서 만들어진 이미지를 타겟으로 다운로드하여 타겟의 하드 디스크에 최종적으로 설치하는 방법이다. 하드 디스크에 설치한 후에는 타겟이 stand-alone 으로 부팅이 가능하다.

1. ‘Install to target system’ 옵션을 선택한다.



2. 메뉴에서 Build > Build All 을 수행한다.
3. 메뉴에서 Build > Deployt Target Image 를 수행한다.
   다음의 파일들이 생성되었는지 확인한다.

   /tftpboot/qplusp.etherboot ← kernel + initrd for Target Installer

   /tftpboot/qplus ← user built kernel to be installed

   /tftpboot/rootfs.tar.gz ← root filesystem to be installed
4. 호스트의 dhcpd 및 tftpd 설정을 확인한다.
   자세한 것은 4.4.1장과 4.4.2 장을 참조
5. 타겟을 부팅한다.
6. 다음과 같은 Target Installer main 화면이 나타날 것이다. 여기에서 '1' 을 눌러 Ethernet install 을 선택한다.

```
1) ethernet install
2) serial install
r) reboot
c) set to defaults
x) start shell
>> 1
```

7. 설정정보와 커널 및 root filesystem 이미지를 받아올 호스트 (여기서는 개발 호스트), 설정 파일의 이름, lilo 의 설치여부를 설정한다.

```
Remote host address: 129.254.180.120
remote host address is 129.254.180.120; is this ok? (y/n/q): y

config name: install.conf
config name is install.conf; is this ok? (y/n/q): y
Getting install configuration
Preparing target device
.
Downloading target root filesystem..
Downloading target kernel..
Run LILO? (y/n): y
..
```

표 1. Target Installer Ethernet install configuration

8. Main 화면에서 ‘R’을 눌러 타겟을 재부팅 한다. 이때 Etherboot 용 floppy 디스크를 제거하고 타겟의 bios 에서 첫번째 booting media 를 하드디스크 혹은 flash disk 로 지정하여 타겟에 설치된 system 으로 부팅할 수 있도록 한다.

4.7.1. 설치 후 update 방법

타겟에서 다음을 수행한다.

```
# /sbin/qp_update
```

자세한 것은 Qplus-P 배포판의 Manuals/remote_update.txt 를 참조한다.

5. Deploy to Target (Arm/Zaurus)

Not Yet Documented..

6. Deploy to Target (Arm/iPAQ)

Not Yet Documented

7. Deploy to Target (Arm/Samsung SDMK2400)

이 장은 삼성의 SMDK2400X (S3C2400 cpu) Boaard 를 기준으로 설명한다. Target Builder 는 SMDK2400X Board Support Package (BSP) 가 설치되어 있어야 한다. 이 장에서는 만들어진 타겟 이미지를 실제로 타겟에서 수행할 수 있도록 Deploy 하는 방법에 대해 설명한다. 또한 부트로더의 설치 및 사용방법도 함께 설명한다.

7.1. Boot Loader

Qplus-P SMDK2400 BSP 에는 Tftp 를 지원하는 boot loader 가 포함되어 있다. 보드에 내장된 부트로더는 고정된 주소에 하나의 이미지를 올리는 것만 가능하기 때문에 제공하는 부트로더를 사용하여야만 linux 를 수행 할 수 있다. 제공하는 부트로더는 보드에 내장되어 있는 부트로더와 같이 사용하는 것도 가능하며 내장된 부트로더 대신 flash 에 설치하여 사용하는 것도 가능하다. BSP 에 포함되어 있는 새로운 부트로더 이미지는 /opt/q+esto/bsp/arm/s3c2400/tools 디렉토리에 있으며 다음과 같다.

netboot-0.5-ram.bin

내장된 부트로더인 24xmon 을 통해 serial 로 다운로드하여 수행할 수 있는 이미지

netboot-0.5-rom.bin

내장된 부트로더를 대신 boot rom 에 flashing 하여 사용할 수 있는 이미지.

제공하는 부트로더는 내장된 부트로더보다 유연하고 강력하며 linux 를 위한 부트로더가 아닌 다른 운영체제의 부트로더로도 사용 가능한다.

NOTICE: netboot 를 사용할 때 하나의 서브넷안에 하나의 타겟만을 사용해야 한다. 현재로서는 EEPROM 으로부터 MAC 주소를 얻어오지 못하기 때문에(EEPROM 이 있는지 확실치 않음) MAC 주소가 고정되어 있다. 이것은 linux 로 부팅한 이후에도 마찬가지이다. 따라서 타겟과 호스트만 허브로 연결하고 외부 네트워크 링크를 차단한 후 사용하는 것을 권장한다.

7.1.1. RAM 에서 netboot 실행하기

우선 새로운 부트로더를 내장되어 있는 부트로더를 유지한 채로 이용하는 방법에 대해 설명한다. 이 방법에서는 새로운 부트로더를 내장된 부트로더를 통해 수행한 후 다시 리눅스 및 기타 운영체제를 올리게 된다. 다음의 순서로 수행한다.

- SMDK2400 보드와 함께 제공되는 dnw.exe 를 실행한다. (실행 후 Configuration/Options 항목에서 Baudrate 과 Com Port 를 115200, Com1 으로 세팅)
- 타겟 보드의 전원을 켠다.
- 타겟의 메모리 테스팅이 끝난 후, dnw 프로그램에서 Serial Port/Transmit 메뉴를 선택한 후 netboot-0.5-ram.bin 을 선택하여, 타겟으로 다운로딩한다.

다운로드가 끝나면 다음과 같은 화면을 볼 수 있다.

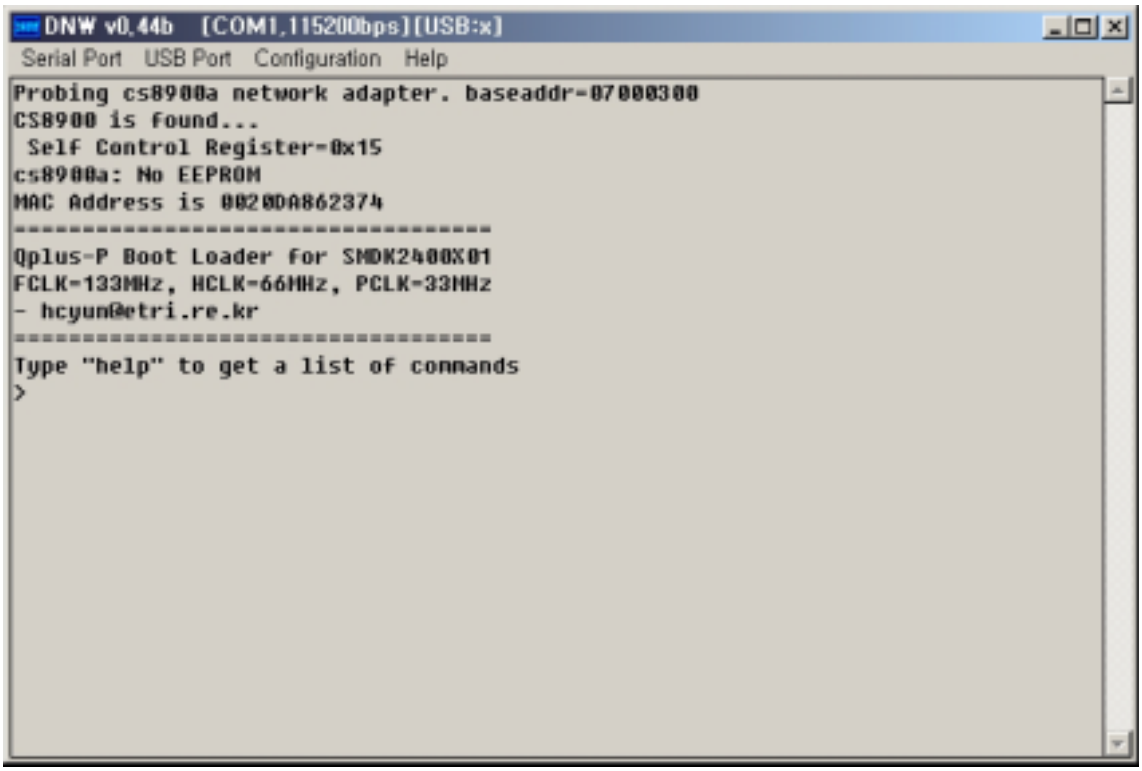


그림 8. bootloader

부트로더에서 실행할 수 있는 명령들은 다음과 같다.

help

```
> help
Help.....
The following commands are supported...
* boot                  Boot Linux with optional kernel options
* tftp                  Download kernel & rootfs to the RAM
* dn <file> <addr>      Download a <file> to <addr>
* jump <addr>           Jump to <addr>
```

```
* setenv <param> <value>   Set environment variable
* printenv                 Print all environment variables
* help                     Get this help
```

boot

형식 : boot

환경변수 ‘kernel-addr’ 에 설정되어 있는 주소로 Jump 한다.
‘jump <kernel-addr>’ 과 동일

tftp

형식 : tftp

환경변수 ‘kernel’, ‘ramdisk’ 에 설정되어 있는 이름의 파일을 각각 ‘kernel-addr’, ‘ramdisk-addr’ 에 설정된 위치로 다운로드함. 다음의 명령들을 수행한 것과 동일함. ‘dn <kernel> <kernel-addr>’ 와 ‘dn <ramdisk> <ramdisk-addr>’ 를 연속하여 수행한 것과 동일함.

dn

형식 : dn <filename> <address>

<filename> 파일을 호스트로부터 <address> 의 주소에 다운로드함.

Jump

형식 : jump <address>

<address> 로 jump 함.

printenv

형식 : printenv

설정된 환경변수를 출력함. 아래는 부트로더의 기본 환경변수 설정이다.

```
param    value
=======================================================
kernel    ss-kernel
```

```
ramdisk  ss-ramdisk
kernel-addr         0x0cf00000
ramdisk-addr        0x0c800000
```

setenv

형식 : setenv <param> <value>

<param> 환경 변수의 값을 <value> 로 세팅

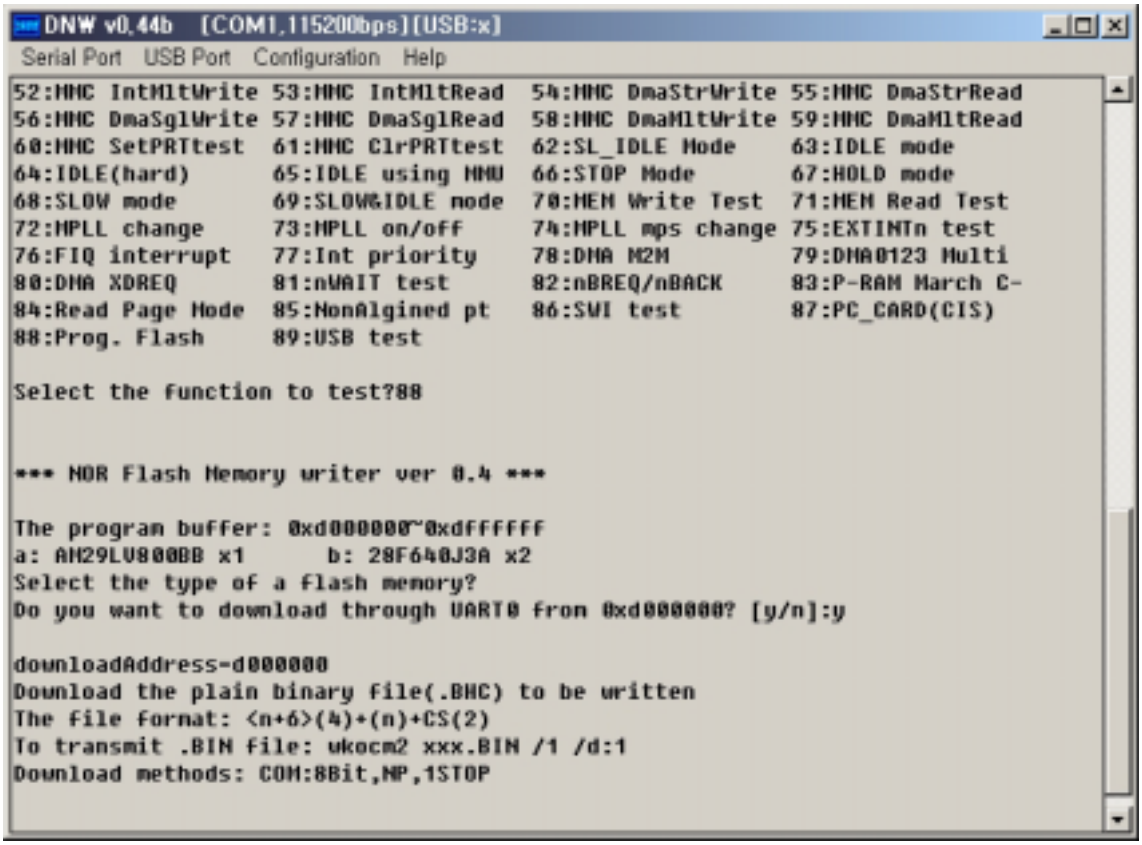
예) > setenv kernel zImage

7.1.2. FLASH 에서 netboot 실행하기 (내장 부트로더 교체방법)

24xtest 의 flash writing 모듈을 이용하여 netboot-0.5.rom.bin 을 flash 에 써서 기본 부트로더로 사용 할 수 있다. BSP 에서 제공하는 netboot 부트롬은 기존의 24xmon 코드를 기반으로 network 를 지원하도록 확장한 것이므로 BSP 에서 제공하는 netboot 부트롬으로 교체하는 것을 권장한다. 기존의 부트로더를 그대로 보존하기를 원한다면 이 절은 건너뛰고 4.1.1 절의 방식으로 사용한다.

Flash 에 굽는 순서는 다음과 같다.

1. 타겟을 부팅한다.
2. DNW 를 이용하여 24xtest.bin 을 다운로드한다.
3. ‘Prog. Flash’ 항목을 선택한다.



4. netboot-0.5-rom.bin 을 다운로드한다.
   0x80000 이 될때까지 계속 writing 을 한다.
5. 타겟을 재부팅 한다.

7.2. Deploy with Initrd root

부트로더를 셋업하였으면 생성한 커널 및 루트 파일시스템 이미지를 타겟에서 수행시킬 수 있도록 Deploy 를 해야 한다.

Initrd 형태로 root 파일시스템의 이미지를 작성한다. Initrd 방식은 root filesystem 이미지가 타겟으로 다운로드되어 ramdisk 에 이미지를 풀어 root filesystem 을 구성한다. 이 방식은 타겟의 메모리가 소요되는 단점이 있지만 별도의 설정이 필요없으므로 테스트가 용이하다. Initrd 방식을 사용하여 target 으로 deploy 하는 순서는 다음과 같다.

1. ‘Use Initrd as a root filesystem’ 옵션을 선택한다.

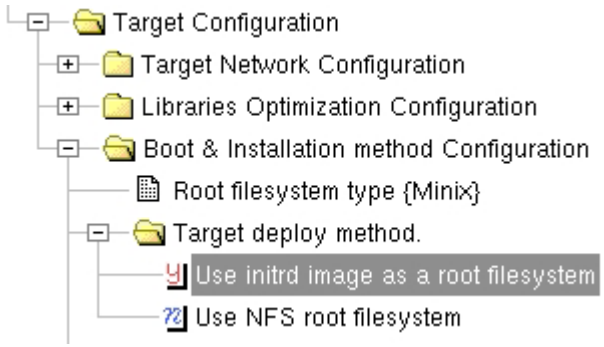


그림 9. Inird deployment option

2. 메뉴에서 Build > Build All 을 선택한다.
   앞서 커널 및 응용 설정에서 Build 를 수행하였다라고 Deploy 옵션에 따라 커널이나 응용 프로그램 설정이 자동으로 바뀔 수 있으므로 마지막으로 한번 더 Build All 을 해 주어야 한다.
   아래 그림은 initrd 를 선택하였을 때 자동으로 체크하는 dependancy 인데 initrd 방식을 선택하였을 때 커널을 'RAM disk support' 와 'Initial RAM disk (initrd) support' 옵션이 활성화 되는 것을 알 수 있다.

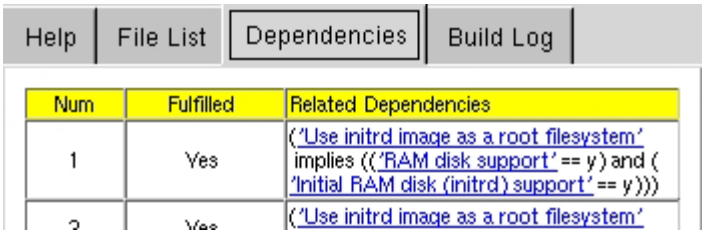


그림 10. Initrd 방식을 선택했을 때 자동으로 체크하는 dependancy

3. 메뉴에서 Build > Deploy Target Image 를 선택한다.
   Intird deploy 에서는 /tftpboot 디렉토리에 다음과 같이 커널 이미지와 ramdisk 이미지가 생성된다.

   /tftpboot/ss-kernel ← kernel image
   /tftpboot/ss-ramdisk ← ramdisk image

그림 11. Initrd deploy log

4. 호스트의 dhcpd 및 tftpd 설정을 확인한다.
   각 서비스의 설정은 4.4 장을 참조한다
5. 타겟을 부팅하여 netboot 부트로더를 수행한다.
   Netboot 를 수행하는 방법은 4.1 장을 참고한다.

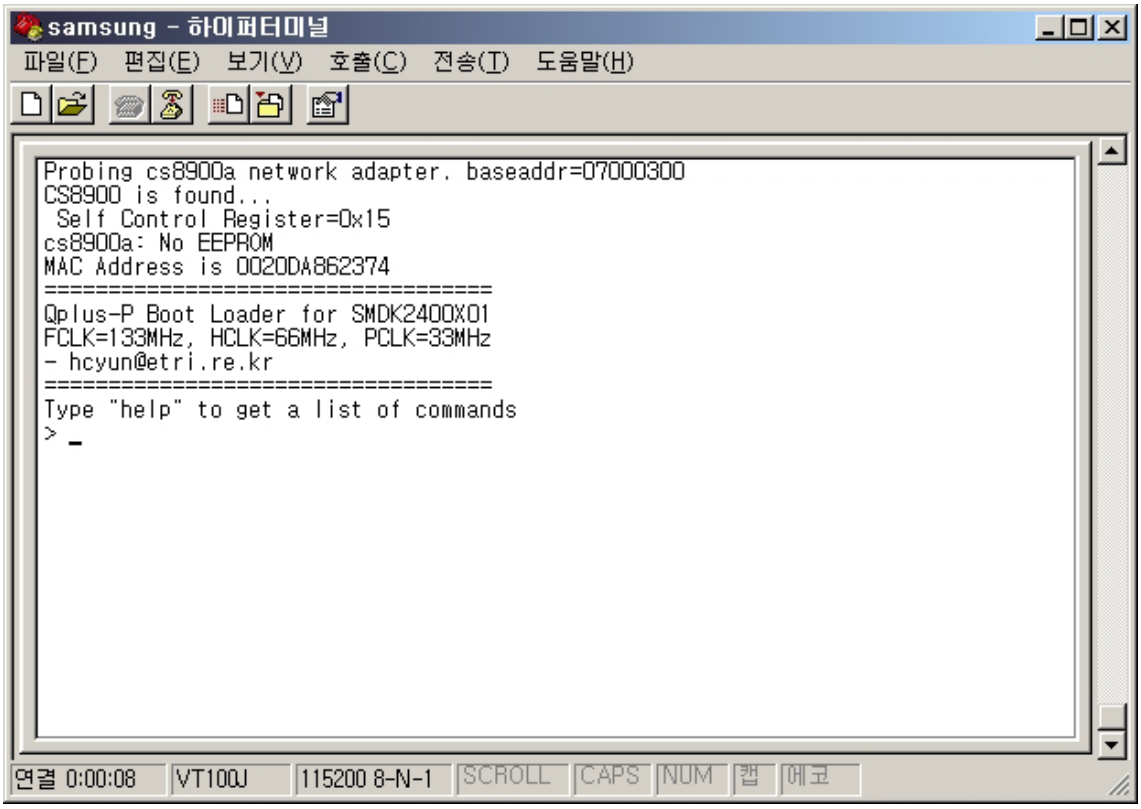


그림 12. netboot 부트로더 초기화면

6. ‘printenv’ 명령으로 커널 및 ramdisk 의 이름 및 다운로드할 address 를 확인한다.

```
> printenv
param   value
================================
kernel  ss-kernel
ramdisk ss-ramdisk
kernel-addr     0x0cf00000
ramdisk-addr    0x0c800000

> _
```

위의 그림에서 화면의 의미는 tftp 를 통해 다운받을 커널의 파일 이름은 ss-kernel, ramdisk 의 이름은 ss-ramdisk 이며 다운받을 주소는 각각 0x0cf00000, 0x0c800000 이라는 것을 나타낸다.

7. ‘tftp’ 명령을 입력한다.

```
-----------------------------------
Type "help" to get a list of commands
> tftp
Sending BOOTP requests . OK
Got BOOTP answer from  129.254.180.120
my address is 129.254.180.119
TFTPing ss-kernel  OK - 576916 Bytes Received
TFTPing ss-ramdisk  OK - 941728 Bytes Received

> _
```

8. ‘boot’ 명령을 입력한다.
   부팅이 성공적으로 수행되면 다음과 같은 Qplus-P 의 초기화면을 만날 수 있다.

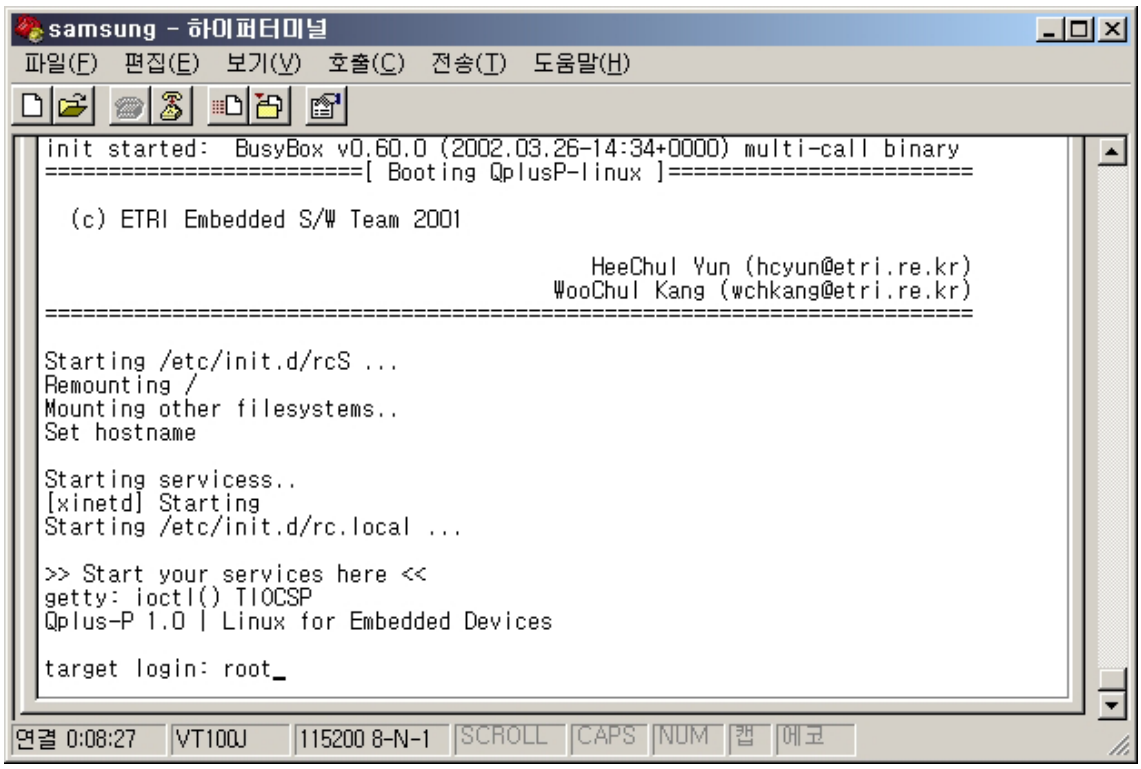


그림 13. 부팅후 Qplus-P 초기화면

7.3. Deploy with NFS root

호스트의 특정 디렉토리를 네트워크를 통해 타겟의 root 로서 사용하는 방법이다. 개발 호스트 상에서 파일들을 마음대로 추가 및 수정이 용이하기 때문에 개발 단계에서 가장 사용하기 적당한 방법이다. SMDK2400 에서도 기본설정으로 이 방식을 사용한다.

1. ‘Use NFS root filesystem’ 옵션을 선택하고 그 하위 옵션 (nfs server 설정)들을 설정한다.

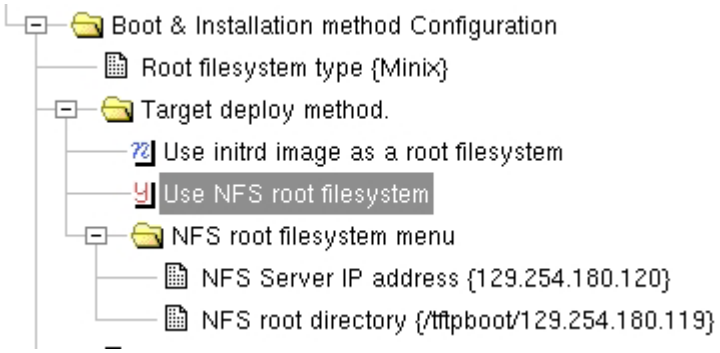


그림 14. nfs deployment option

'Use NFS root filesystem' 옵션을 선택하면 그림에서 보는 것과 같이 두개의 하위 옵션이 나타난다. 'NFS Server IP address' 는 nfs server (일반적으로 target builder 를 수행하는 호스트와 동일한 호스트)의 ip address 를 입력하며, 'NFS root directory' 옵션은 타겟이 root 디렉토리로 사용할 호스트의 디렉토리 이름을 입력한다. 여기에서 설정한 정보는 타겟 부팅시 nfs server 에 대한 기본 설정 정보로 이용된다.

2. 메뉴에서 Build > Build All 을 선택한다.
   nfs deploy 은 kernel 의 NFS 설정과 응용 프로그램 중 chroot, init 등과 dependancy 가 존재한다. 따라서 Deploy 전에 Build All 을 선택하여 이러한 dependency 관계에 의한 변경사항이 자동으로 반영되도록 한다.

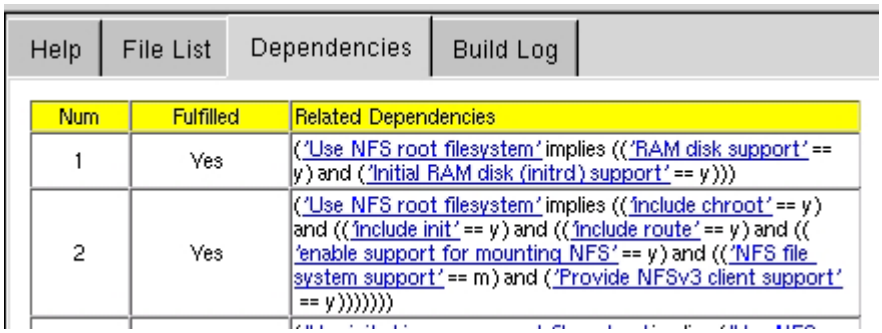


그림 15. nfs deploy dependancy

3. 메뉴에서 Build > Deploy Target Image 를 선택한다.
   Nfs deploy 에서는 /tftpboot 디렉토리에 다음의 파일들이 생성된다.

/tftpboot/ss-kernel	← kernel image
/tftpboot/ss-ramdisk	← nfs boot 용 initrd 이미지
/tftpboot/rootfs.tar.gz	← target root 파일시스템 아카이브

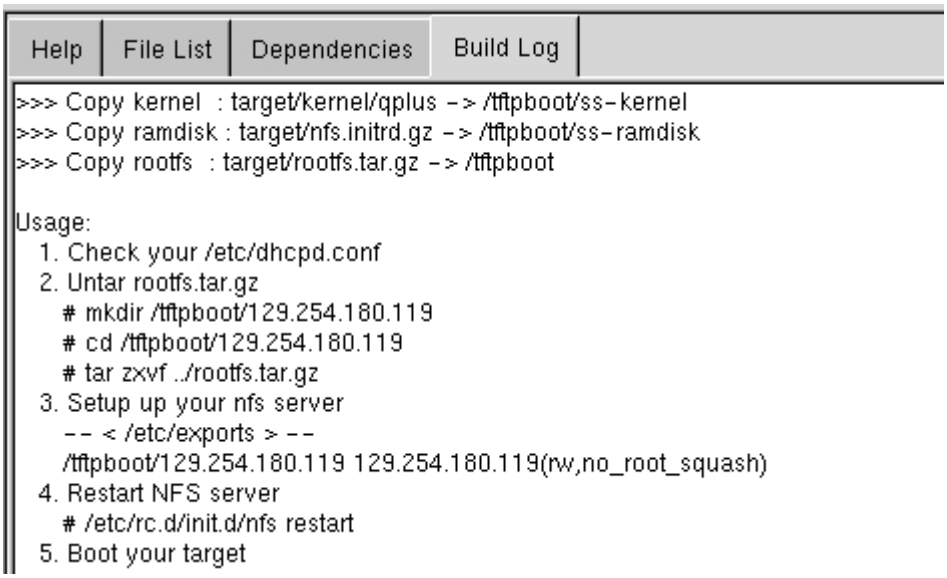


그림 16. nfs deploy 시의 log 화면

위의 그림은 deploy 시의 log 화면이다. 사용자는 이 화면에 나타나 있는 네트워크 주소등의 정보는 모두 Target Builder 에서 사용자가 설정한 정보를 반영하기 때문에 그대로따라하면 되도록 되어 있다.

4. 생성된 rootfs.tar.gz 를 설치한다.
   rootfs.tar.gz 타겟의 root 파일시스템의 내용을 가지고 있는 압축파일로서 사용자는 그림 8 의 Usage 의 2 번 항목을 그대로 따라하면 된다.

```
# mkdir /tftpboot/<target root dir>
# cd /tftpboot/<target root dir>
# tar zxvf ../rootfs.tar.gz
```

5. NFS 서버 설정을 확인한다.
   그림 8. 의 3 번 항목에 있는 것과 같이 /etc/exports 에 호스트에 설치한 타겟의 root 파일시스템 디렉토리가 export 되었는지 확인한다. 만일 /etc/exports 파일을 수정하였다면 다음과 같이 NFS 서버를 재시동 한다.

```
# /etc/rc.d/init.d/nfs stop
# /etc/rc.d/init.d/nfs start
```

   자세한 것은 4.4 장을 참고한다.

6. 타겟을 부팅하여 netboot 부트로더를 수행한다.
   'Deploy with Initrd' 에서의 설명과 동일
7. 'tftp' 명령을 입력한다.
   'Deploy with Initrd' 에서의 설명과 동일
8. 'boot' 명령을 입력한다.
   부팅을 하게 되면 다음과 같이 사용자의 NFS 서버 설정을 묻는 화면이 나타난다.

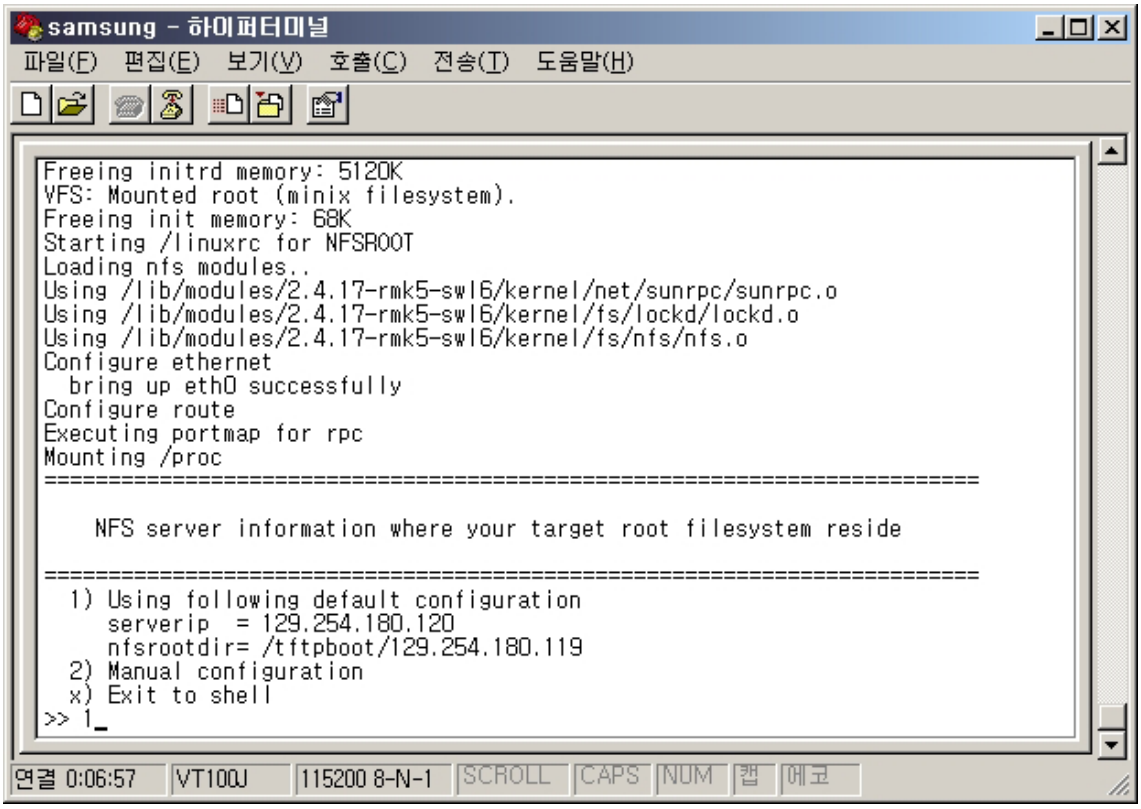


그림 17. NFS 부팅시의 Dialog 화면

이 화면에서 1) 을 선택하면 default 설정 (Target Builder 의 NFS Server 설정 항목의 정보) 을 이용하며, 2)를 선택하면 사용자가 매뉴얼하게 서버 설정을 할 수 있다. 3) 을 선택하면 Initrd 의 shell 로 빠져나갈 수 있다.

7.4. 호스트 서비스 설정

netboot 를 사용하기 위해서는 dhcpd (또는 bootpd) 와 tftp 서버가 호스트에서 설정되어야 한다.

각각은 4.4.1장과 4.4.2장을 참고한다.

8. 일반 응용프로그램을 프로젝트에 추가하기

현재 Qplus-P 타겟 빌더는 약 30 개 정도의 기본 응용 프로그램 패키지를 제공한다. 그러나, 이러한 패키지들은 리눅스가 동작하는데 필요한 기본적인 패키지 일뿐 실제 임베디드 시스템 개발과정은 더욱 다양한 응용을 Qplus-P 타겟 빌더 상에서 설정하고, 빌드해서 , 타겟에 적재해보기를 요구한다. Qplus-P 는 이러한 요구를 다음과 같이 두가지 방법으로 지원한다.

- ***첫번째 방법***은 개발된 응용 프로그램을 단순히 타겟 빌더가 생성한 루트 파일시스템에 합치는 방법이다. 이는 타겟 빌더를 이용해서 설정을 할 수는 없고 단지 최종적으로 생성될 타겟 루트 파일시스템에 원하는 응용 프로그램을 합치는 기능이다.
- ***두번째 방법***은 타겟 빌더의 기본 패키지와 마찬가지로 타겟 빌더에 하나의 응용 패키지로써 통합시키는 방법이다. 이는 기본 패키지와 마찬가지로 타겟빌더 상에서 설정과 빌드 등을 할 수 있게 된다.

이번 장에서는 위에서 언급한 두가지 방법으로 응용프로그램을 타겟 빌더에 추가하는 것에 대해서 각각 설명한다.

8.1. 파일을 타겟 이미지에 합치기

타겟 빌더는 설정에 의해 선택된 아이템들로 <프로젝트 디렉토리>/***target/rootfs*** 디렉토리에 임시 타겟 루트파일 시스템을 생성한다. 이 임시 루트 파일 시스템은 최종적으로 타겟에 적재될 형태로 변형되어 설치되는 것이다.
타겟 빌더는 사용자가 작성한 응용 프로그램등을 이 임시 루트 파일 시스템의 내용과 합칠 수 있는 방법을 제공한다. 다음은 이를 위한 순서이다.

1. **<프로젝트 디렉토이>/target/usersupp 디렉토리에 파일을 위치시킨다.**

타겟 루트 파일 시스템에 합치기를 원하는 파일들을 <***프로젝트 디렉토리***>***/target/usersupp*** 에 위치 시킨다. 여기서 주의 할 것은 타겟 빌더는 <***프로젝트 디렉토리***>***/target/usersupp*** 디렉토리를 실제 타겟 시스템의 루트 디렉토리로 해석한다는 것이다. 예를 들어 ***test.sh*** 라는 파일이 타겟의 ***/usr/bin*** 에 설치되기를 원한다면 <***프로젝트 디렉토리***>***/ target/usersupp/usr/bin/***에 ***test.sh*** 파일을 위치시켜야 한다.

2. 루트 파일 시스템의 생성

메뉴의 ***Build*** > ***Build root filesystem*** 또는 ***Build*** > ***All*** 를 통해 루트 파일 시스템을 생성할 때 프로젝트***/target/usersupp*** 디렉토리의 파일들이 ***target/rootfs*** 에 복사되어 들어간다.

주의 : 타겟루트 파일 시스템에 실제 파일들이 적재될 때는 소유권과 퍼미션이 그대로 전달됨으로 이에 주의 한다. 즉, 루트권한 이여야만 하는 파일등의 경우 루트 권한으로 설정해 두어야 한다..

8.2. 타겟 빌더 패키지 만들기

앞 서의 방법은 일부 파일을 타겟에 적재하고 싶은 경우에는 쉽게 사용될 수 있는 방법이지만, 타겟 빌더의 프로젝트 상태에서 설정하고 빌드하는 과정을 전혀 활용할 수 없는 단점을 가진다.

타겟 빌더의 패키지로써 포함시키면 내장된 커널/응용 패키지와 함께 설정을 할 수 있게 된다.

8.2.1. 패키지의 구성 요소

각각의 타겟 빌더 패키지는 다음과 같은 파일들로 구성되어 있다.

파일	용도
SRPM 파일	패키지의 tar.gz 압축형식의 소스 파일과 이의 컴파일 인스톨에 관여하는 spec 파일로 구성됨
QPD 파일	타겟 빌더가 사용하기 위한 정보를 기술하는 파일

SRPM 파일은 압축된 소스와 이를 컴파일하고 인스톨 하는 방법을 기술한 파일을 가지고 있는 것으로 rpm(redhat package manager)에 의해 생성된다. SRPM 을 생성하기 위해서는 spec 파일과 압축된 소스파일을 지정된 위치(Redhat Linux 의 경우 /usr/src/redhat)에 놓고 “***rpm –ba*** <***패키지의 spec 파일***>” 명령을 수행하면 된다.

알림: rpm 에 대한 좀더 자세한 설명은 http://www.rpm.org/max-rpm 문서를 참조하기 바란다.

QPD(Qplus Package Descriptor)파일은 타겟 빌더가 각 패키지에 대한 정보를 얻기 위해서 사용하는 파일이다. 패키지를 추가하기 위해서는 각 패키지의 설정 가능한 사항들에 대해서 이 파일에 기술해야 타겟 빌더 상에 표시되어 나타난다. 대부분의 패키지는 간단한 QPD 파일 만으로도 충분하며, 이는 쉽게 각 패키지의 spec 파일에 첨부되어 작성될 수 있다.

8.2.2. QPD 파일 작성하기

QPD 파일은 각 패키지의 rpm spec 파일의 끝에 추가적인 패키지 정보를 기술한 파일이다. 즉,

QPD 파일 = rpm spec 파일 + 추가 패키지 정보

이다.

QPD 파일의 문법

다음은 QPD 파일에 패키지 정보를 기술할 때의 syntax 이다. QPD 파일은 아래의 규칙에 맞게 작성되어야 한다.

```
QPD ::= <일반 스펙 파일 부분>  <패키지 정보>

;; QPD 파일의 시작은 ' %package'  태그로부터 시작됨
;; 패키지 정보는 옵션들의 연속이다.
<패키지 정보>  ::=  ' %package'   <패키지 이름>
                    <옵션 서브필드>
                     [<옵션>  | <그룹>]*

;; 각 옵션은 이름과 성질을 가짐
<옵션>  ::=  ' %option'  <옵션 이름>
             < 옵션 서브 필드>

;; 그룹은 간단한 설명만 가지는 옵션이다.
<그룹>  ::=' %group'   <그룹 이름>
             <그룹 서브 필드>

;; 옵션의 각 서브 필드들은 옵션이 가지는 성질을 말함
<옵션 서브 필드> ::==  <prompt>
                       [<export symbol>]
                       [<files>]
                       [<require>]
                       [<provide>]
                       [<build_vars>]
                       [<help>]

<그룹 서브 필드> ::== <prompt>
```

```
                              [<help>]
;; 옵션에 대한 간단한 설명
<prompt> ::= ‘  %%prompt’     <string>

;; 옵션에 대한 자세한 설명
<help> ::= ‘  %%desc’     <string>

<build_vars> ::= ‘  %%build_vars’   <string>

;; 옵션이 선택된 경우 외부 헤데 파일에 기록할 매크로심벌들
<export symbol> ::= ‘  %%export_symbol’     <external symbol> *

;; 옵션이 선택될 경우 타겟에 깔릴 파일들의 리스트
<files> ::= ‘  %%files’     <file name>*

;;옵션의 의존성을 여기 기술함
<require> ::= <logical>

;; 옵션간의 충돌을 없애기 위함
<provide> :: = ‘  %%provide’     <symbol>*

<옵션이름>, <패키지 이름> :: = [A-Za-z][A-Za-z0-9/]*
<symbol> ::= [A-Za-z0-9_-]*
<string> ::= '[^']*'|"[^"]*";
<decimal>       ::= [0-9]+
<hexadecimal> ::= 0x[A-Fa-f0-9]+
<tritval>       ::= [ymn]

<expr> ::= <expr> '+' <expr>
          | <expr> '-' <expr>
          | <expr> '*' <expr>
          | <ternary>

<ternary> ::= <expr> '?' <expr> ':' <expr>
              | <logical>

<logical> ::= <logical> 'or' <logical>
              | <logical> 'and' <logical>
              | <logical> 'implies' <logical>
              | <relational>

<relational> ::= <term> '==' <term>
                 | <term> '!=' <term>
                 | <term> '<=' <term>
                 | <term> '>=' <term>
                 | <term> '>' <term>
                 | <term> '<' <term>
                 | <term>
                 | 'not' <relational>
```

```
<term> ::= <term> '|' <term>      ;; maximum or sum or union value
         | <term> '&' <term>      ;; minimum or multiple or intersection value
         | <term> '$' <term>      ;; similarity value
         | <atom>

<constant> ::= <tritval>
             | <string>
             | <decimal>
             | <hexadecimal>

<atom> ::= <symbol>
         | <constant>
         | '(' <expr> ')'
```

패키지(%package), 그룹(%group) , 옵션(%option)

QPD 파일은 각 응용 패키지의 성질을 계층적으로 기술하기 위해 %package , %group , %option 아이템을 가진다.

%package <***패키지 이름***> 아이템은 QPD 고유영역의 시작이며 패키지의 전체적인 성질을 기술하기 위해서 사용되며 하나의 QPD 파일마다 한 번만 사용될 수 있다. ***%option*** <***옵션이름***> 아이템은 패키지내의 각 설정 가능한 부분을 옵션화하기 위해 사용되며 다양한 속성을 가질 수 있다. ***%group*** <***그룹이름***> 아이템은 %option 태그의 일종으로 %option 들을 그룹핑하기 위한 용도로 사용되며 다양한 속성을 가질수 없고 단지 설명을 위한 %%prompt, %%desc 속성만을 가진다.

계층 구조와 이름 명명 규칙

패키지, 그룹, 옵션은 계층적인 모습을 가지며 한 패키지 내에 옵션과 그룹이 존재하며, 옵션과 그룹은 그 안에 또 다른 그룹과 옵션을 재귀적으로 가질 수 있다. 이러한 재귀적 구조는 타겟 빌더 상에서 트리 형태로 표현되어 나타나게 된다.

그룹과 옵션은 자신이 다른 옵션이나 그룹, 패키지에 속하는 계층구조를 이름을 이용해서 표현한다. 자신이 다른 옵션이나 그룹 패키지에 속함을 표현하기 위해 자신의 이름을 만들 때 자신이 속한 아이템의 이름을 재귀적으로 앞에 붙이고 각 아이템 이름 사이에는 “/”를 넣어서 구별한다. 예를 들어 foo 라는 패키지내에 goo 라는 옵션이 있고 goo 옵션은 hoo 라는 옵션을 포함할 경우 다음과 같이 표현이 가능하다.

```
%package foo            <= 패키지 foo
…
%option foo/goo         <= 옵션 goo 는 foo 패키지에 속한 옵션이다
```

…

%optin foo/goo/hoo <= 옵션 hoo 는 goo 옵션에 속하는 옵션이다.

주의: 패키지,옵션,그룹의 이름은 알파벳과 숫자만이 사용가능하고, 대소문자 구별을 하지 않음을 주의 하라. 이는 내부적으로 사용되는 CML2 엔진이 그외의 문자를 쓸수 없도록 강제하기 때문이다.

속성들

각 아이템(패키지, 옵션, 그룹)들은 자신의 속성을 가지는데 이는 해당 아이템이 타겟 빌더의 설정 창에 보여줄 정보와 선택된 경우 일어날 행동을 기술하는 것이다.
아이템이 가질수 있는 속성은 다음과 같다.

이름	설명	적용 가능 아이템
%%prompt	아이템에 대한 짧은 설명. 타겟빌더 트리상에 보여지는 아이템의 이름은 여기에 입력된 짧은 설명이 보여지며 실제 아이템의 이름은 내부적으로만 사용된다.	모두
%%desc	아이템에 대한 긴 설명	모두
%%files	아이템이 선택된 경우 타겟 시스템에 깔릴 파일들의 리스트. 여러 개의 파일이나 디렉토리를 나열할 수 있다.	패키지, 옵션
%%require	아이템이 요구하는 의존성 규칙을 설정한다.	패키지, 옵션
%%provide	아이템간의 충돌을 회피하기 위함. 같은 심벌을 provide 하는 아이템들은 동시에 선택될 수 없다.	패키지, 옵션
%%export_symbol	매프로 심벌로 응용 프로그램의 컴파일과 기능을 제어하는 일부의 응용(busybox, tinylogin)등을 위해 필요하다.	패키지, 옵션
%%build_vars	아이템이 선택된 경우 빌드 과정에서 옵션등을 특정 문자열로 교체한다.	패키지, 옵션

속성은 각 아이템에 덛붙여 기술하면 된다. 속성들간의 순서는 없다.
예를 들어 foo/goo 라는 옵션이 %%prompt, %%desc, %%files, %%require 속성을 가지는 경우 다음과 같이 기술한다.

```
%option      foo/goo                 <=  foo 패키지에 속한 goo 옵션
%%prompt     some special option      <=  간단한 설명
%%files      /usr/bin/good-file      <=3 개 파일이 타겟에 인스톨될 것임
             /usr/bin/bad-file
             /etc/goo.conf
%%require    foo/hoo ==y             <= foo/hoo 옵션이 선택되어야만 됨
%%desc                               <= 매우매우매우~~~ 긴 설명
it is a very very very long long long
```

```
long~~~~~long help file.
```

파일 리스트의 기술

파일리스트 속성은 각 패키지/옵션 아이템이 선택되었을 때 타겟에 인스톨 될 파일의 리스트를 기술하는 것이다.

예를 들어 foo 라는 패키지가 선택되면 기본적으로 /usr/local/bin/prog1, /usr/local/bin/prog2 라는 파일이 인스톨 되어야 하고, /usr/local/bin/prog3 은 선택적으로 깔수도 있고 그렇지 않을 수도 있다면 다음과 같이 사용할 수 있다.

```
...
%package        foo
%%prompt        good package
%%files         /usr/local/bin/prog1
                /usr/local/bin/prog2
...
%option         foo/hoo
%%files         /usr/local/bin/prog3
```

의존성 규칙의 기술

각 패키지의 아이템들은 자신이 가지는 의존성을 %%require 속성으로 기술 할 수 있다. 의존성은 다른 옵션이나 커널 심벌과의 관계를 표현하는 것으로, 타겟 빌더는 각 아이템의 값을 설정 할 때마다 관련 의존성을 모두 만족하는 지를 검사해서 그렇지 않을 경우 경고 메시지를 보여주고 설정값을 원래대로 되돌리는 일을 한다.

%%require 은 기본적으로 논리식으로 의존성 규칙을 표현하며 다음과 같은 논리 연산자가 사용가능하다.

and

or

not:

== , *!=*, > , < , >= , <=

Implies (포함 관계를 나타낸다. ⊃ 또는 →와 같은 의미임)

다른 옵션을 지정하기 위해서는 해당 옵션의 절대패스 이름(자신의 부모를 재귀적으로 나타낸 이름)을 사용해서 나타낸다. 또한 각 옵션이 선택된 상태는 y 로, 선택되지 않은 상태는 n 으로 표시하며 m 은 모듈로써 선택되었음을 표시한다.

예를 들어 foo 패키지내의 foo/goo/hoo 라는 옵션이 haa 패키지의 haa/huu/hee 라는 옵션이 선택되고 haa/huu/hii 옵션은 선택되지 않기를 원한다면 다음과 같이 기술 가능하다.

```
%option foo/goo/hoo
%%require haa/huu/hee ==y and haa/huu/hii ==n
```

위와 같이 의존성이 기술된 경우 타겟 빌더 상에서 foo/hoo/hoo 옵션을 Enable 상태로 만족시키기 위해서는 먼저 haa/huu/hee 옵션은 Enable 상태로 haa/huu/hee 옵션은 Disable 상태로 만들어 줘야한다.

옵션간의 충돌 예방

옵션이나 패키지 중에는 동시에 같이 선택되어서는 안되는 경우도 있다. 예를 들어 ps 명령어는 busybox 패키지에서 제공하는 것을 사용할 수도 있고, procps 패키지에서 제공하는 것을 사용할 수도 있지만 두개가 동시에 같은 위치에 깔릴수는 없다. 이를 방지하기 위해서 앞서 말한 의존성 규칙을 사용할 수도 있지만 타겟 빌더는 이를 간단히 해주는 방법을 %%provide 속성을 통해 제공한다.

%%provide 문장에서 같은 심벌을 provide 하는 옵션/패키지는 동시에 같이 선택될 수 없다. 심벌은 알파벳과 숫자로 구성된 임의의 심벌이며, 다른 아이템과 중복되지 않도록 주의해야 한다. 앞서의 ps 명령어의 예를 QPD 파일에는 다음과 같이 기술 할 수 있다.

busybox.qpd

```
. . .
%package busybox
. . .

%option busybox/ps
%%provide PS
. . .
```

procps.qpd

```
. . .
%package procps
. . .
%option procps/ps
%%provide PS
. . .
```

위에서 두 옵션은 PS 라는 동일한 심벌을 provide 하므로 동시에 선택될 수 없다.

빌드 옵션의 제어

타겟 빌더는 각 패키지의 SRPM 을 컴파일 할 때 SRPM 내에 들어 있는 spec 파일을 쓰지않고 qpd 파일내의 spec 영역에 기술된 정보에 따라 컴파일 하게 된다. 따라서 spec 영역의 컴파일 관련 옵션을 타겟빌더 설정 상태에 따라 제어할 수 있다.

%%build_vars 는 자신이 속한 아이템이 선택될 경우 spec 영역의 특정 문자열을 대치시킨다. “!!” 기호를 대치시킬 문자열의 양끝에 삽입하고 %%build_vars 에는 해당 문자열을 무었으로 대치시킬 지를 기술해야 한다.

예를 들어 foo 라는 패키지가 정적으로 컴파일 되기 위해서는 make 의 옵션으로 “DOSTATIC=true”라는 옵션을 줌으로써 가능하다고 하자. 우선, qpd 화일의 %build 섹션에 기술된 make 문장을 다음과 같이 수정한다.

```
. . .
%build
make !!MYOPT!!
. . .
```

그리고 다음과 같이 %%build_vars 필드를 사용해서 대체될 문자열을 무었으로 대체할 지를 기술한다.

```
. . .
%option tinylogin/static
%%prompt: static compilation?
%%build_vars: MYOPT=”DOSTATIC=true”
. . .
```

이는 tinylogin 패키지의 tinylogin/static 옵션이 선택될 경우 %build 섹션에 기술된 !!MYOPT!! 문자열을 DOSATIC=true 로 대체시키게 된다.

QPD 내의 spec 영역의 작성

Spec 영역은 rpm spec 파일에서 사용되는 방식을 그대로 사용할 수 있다. 다만 다음과 같은 제약 사항을 가지므로 유의해야 한다.

- **QPD 파일의 spec 영역에는 서브 패키지 개념을 사용할 수 없다.**

  RPM 은 %package 태그를 사용해서 서브 패키지를 가질수 있다. 그러나 이는 QPD 파일내에서 QPD 영역의 시작을 표시한 %package 태그와 중복된다. 또한 QPD 는 아직 서브 패키징의 개념을 지원하지 않는다. 따라서 서브 패키징이 되어있는 SRPM 패키지는 서브 패키지를 생성하지 않도록 수정해야 한다.

- **매크로 환경 변수 사용의 제약**

  QPD 의 스펙영역중 Preamble 부분의 Name, Group, Version 에 있는 내용과 QPD 영역의 내용은 rpm 변수에 대한 처리를 하지 않는다. 즉 , Name 이 foo 라고 하면 %{Name} 이라는 변수는 foo 로 해석되지 않고 %{Name}으로 그냥 사용된다. 따라서 주의를 요한다.

8.2.3. SRPM 과 QPD 파일의 등록

프로젝트 디렉토리밑의 package 디렉토리는 각종 패키지가 놓여지는 곳이다. SRPM 디렉토리에 작성된 SRPM 파일을 , QPDS 디렉토리에 QPD 파일을 위치시킨다.
위와같이 SRPM 파일과 QPD 파일을 배치시키고 타겟 빌더를 수행시키면 새로 추가된 패키지가 반영되어 나타난다.

8.2.4. 패키징 예제

여기서는 간단한 패키지의 QPD 파일을 작성하고 등록하는 과정을 설명한다.
좀더 많은 예제를 위해서는 <프로젝트 디렉토리>/packages 를 참조하기 바란다.

foo 패키지를 추가 한다고 가정할 때 다음과 같은 순서로 진행된다.

SRPM 파일의 작성

- foo-1.0.tar.gz 파일을 /usr/src/redhat/SOURCES 디렉토리에 배치한다.
- Spec 파일을 작성해서 /usr/src/redhat/SPECS 디렉토리에 배치한다.
- 루트 권한 으로써 다음 명령어를 수행한다.

  rpm –ba foo.spec

- /usr/src/redhat/SRPMS 디렉토리에 foo-1.0.src.rpm 이 생성되었다.

QPD 파일의 작성

- 위에서 사용된 spec 파일을 복사해서 foo.qpd 파일로 확장자를 수정한다. 다음은 이 파일을 보인다.

```
Summary: foo is a very simple and nice program
```

```
Name: foo
Version: 1.0
Release: 1
Copyright: GPL
Group: System Environmanet/Base
Source: ftp://ftp.etri.re.kr/foo-1.0.tar.gz
BuildRoot: /var/tmp/%{name}-buildroot

%description
Do you nedd more explaination about this famous package

%prep
%setup -q

%build
make

%install
rm -rf $RPM_BUILD_ROOT
make PREFIX="$RPM_BUILD_ROOT" install

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-, root, root)
/
```

먼저, 위 QPD 파일의 하단에 %package 아이템을 넣는다.

```
. . .
%package foo
%%prompt foo
%%files    /usr/local/bin/hoo
           /usr/local/bin/haa
%%require goo/gee == y
%%desc
foo is a very import package which has
many good functions
. . .
```

위는 foo 패키지가 설치될 경우 두개의 파일이 설치 될 것이고, goo 패키지의 goo/gee 옵션이 Enable 되어야 된다는 것을 요구한다.

/usr/local/hee 라는 파일은 선택적으로 깔리게 하고 싶은 경우 다음과 같이 옵션을 추가한다.

```
. . .
%package foo
%%prompt foo
%%files    /usr/local/bin/hoo
```

```
                /usr/local/bin/haa
%%require goo/gee == y
%%desc
foo is a very import package which has
many good functions

%%option foo/hee
%%prompt include hee command ?
%%files     /usr/local/hee
%%desc
hee is a some nice file .
but you can select it optionally~~~
. . .
```

최종적으로 생성된 QPD 파일은 다음과 같다. 굵은 글씨로 되어 있는 부분이 QPD 에서 설정아이템을 기술하기 위해 추가된 부분이다.

```
Summary: foo is a very simple and nice program
Name: foo
Version: 1.0
Release: 1
Copyright: GPL
Group: System Environmanet/Base
Source: ftp://ftp.etri.re.kr/foo-1.0.tar.gz
BuildRoot: /var/tmp/%{name}-buildroot

%description
Do you nedd more explaination about his famous package

%prep
%setup -q

%build
make

%install
rm -rf $RPM_BUILD_ROOT
make PREFIX="$RPM_BUILD_ROOT" install

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-, root, root)
/


%package foo
%%prompt foo
%%files    /usr/local/bin/hoo
           /usr/local/bin/haa
%%require goo/gee == y
%%desc
```

```
foo is a very import package which has
many good functions

%%option foo/hee
%%prompt include hee command ?
%%files     /usr/local/hee
%%desc
hee is a some nice file .
but you can select it optionally~~~
```

QPD 와 SRPM 파일의 배치

- <프로젝트 디렉토리>/packages/QPDS 에 foo.qpd 를 복사한다.
- <프로젝트 디렉토리>/packages/SRPMS 에 foo-1.0.src.rpm 을 복사한다.

타겟 빌더를 새로 수행시킨다.

타겟 빌더 상에는 추가된 패키지가 표시된다.

9. 터미널에서의 타겟 빌더 사용법

타겟 빌더 GUI 고유의 기능 (filelist, dependency display) 을 제외한 기본적인 모든 기능은 쉘 프롬프트상에서도 가능하다. 즉 프로젝트 생성부터 타겟 시스템에 적재하기까지의 모든 과정을 쉘 상에서 명령어 방식으로 수행 가능하다. 이는 시스템 전체가 매우 유연하게 설계되었음을 보여준다. 타겟 빌더의 전체 시스템 설계는 별도의 시스템 설계 문서를 참조하기 바란다.

GUI 의 사용을 좋아하지 않거나 시스템 성능등의 문제로 인해 GUI 를 사용하는 것이 어렵다면 본 장을 참고하기 바란다.

9.1. 프로젝트 생성

그림 18. 프로젝트 생성

1. 프로젝트 이름 입력

   프로젝트의 이름을 입력한다.

   ```
   1. Enter your project name
   => ttt
   ```

2. 아키텍쳐 입력

   화면에 출력된 아키텍쳐의 목록중 원하는 것을 입력한다.

   ```
   2. Enter your target architecture name
      Available architectures : arm i386
   => arm
   ```

3. 보드 이름 입력

화면에 출력된 보드의 목록중 한가지를 입력한다.

```
3. Enter your target board name
   Available boards : s3c2400
=> s3c2400
```

여기까지 입력하면 다음과 같이 프로젝트를 생성하고 메시지를 출력하여 준다.

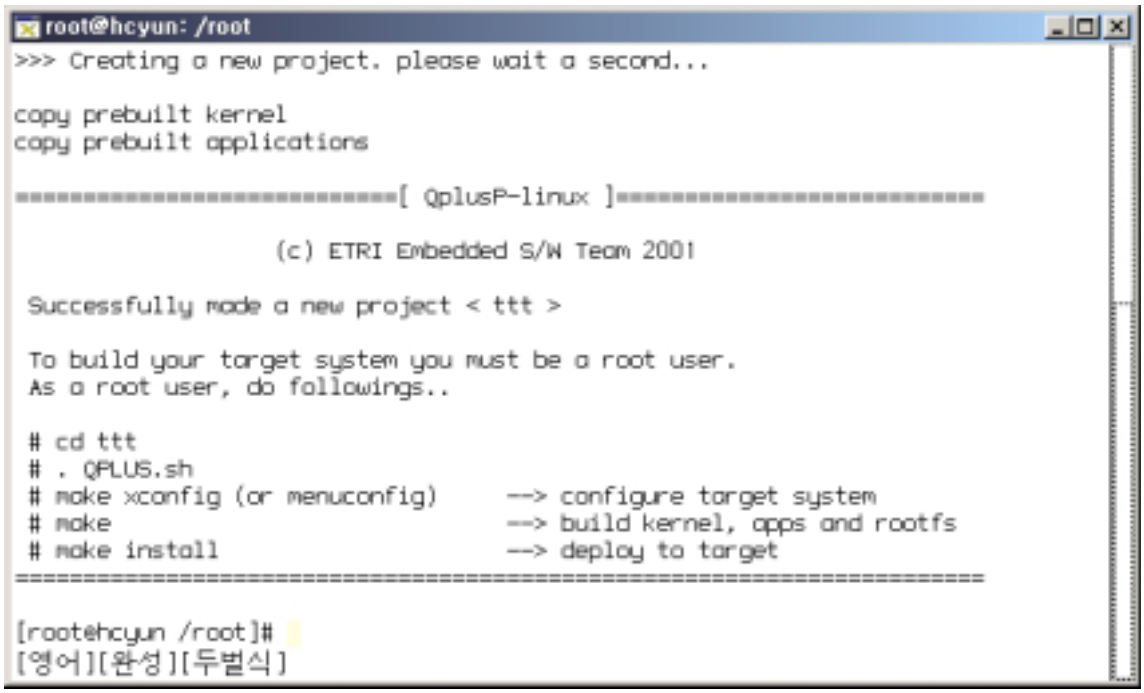


그림 19. 프로젝트 생성후의 초기화면

4. QPLUS.sh 수행하기

환경변수를 로딩한다. 다음을 수행한다.

```
# . QPLUS.sh
또는
# source QPLUS.sh
```

9.2. 시스템 설정

모든 시스템 설정은 CML2 rule 에 의해 기술되며 따라서 어떠한 CML2 configurator 로도

설정을 하는 것이 가능하다. 타겟 빌더 GUI 도 기본적으로 CML2 Configurator 이며 그외에 추가적인 기능이 포함된 것이다. 여기에서는 CML2 의 기본 설정기인 menuconfig 과 xconfig configurator 를 기준으로 설명한다. 어떠한 configurator 를 사용하더라도 모든 설정은 결국 프로젝트 디렉토리에 위치한 config.out 파일에 기록된다.

9.2.1. menuconfig

CML2 의 menuconfig configurator 의 모습니다. 타겟 빌더를 사용하였을때의 설정항목과 동일한 설정항목을 보여준다. 타겟 빌더 사용자 가이드에 있는 설정항목에 대한 설명을 참고하여 커널, 시스템, 타겟 설정을 수행한다.

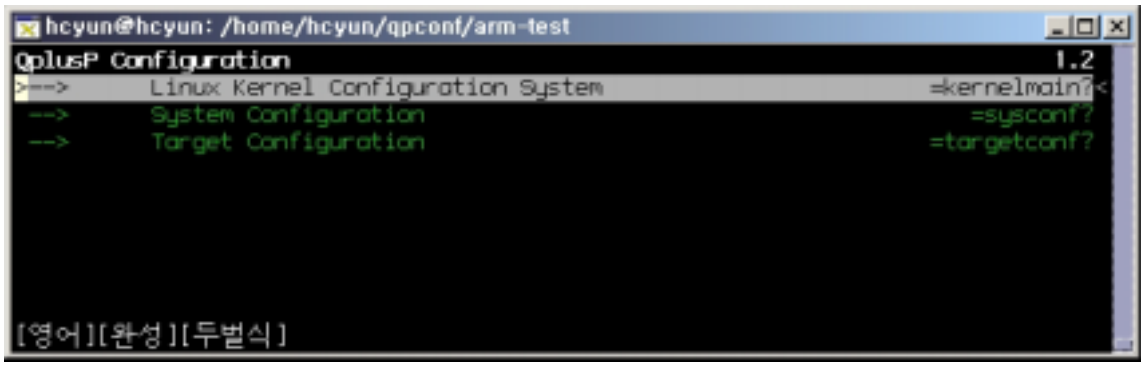


그림 20. Menuconfig

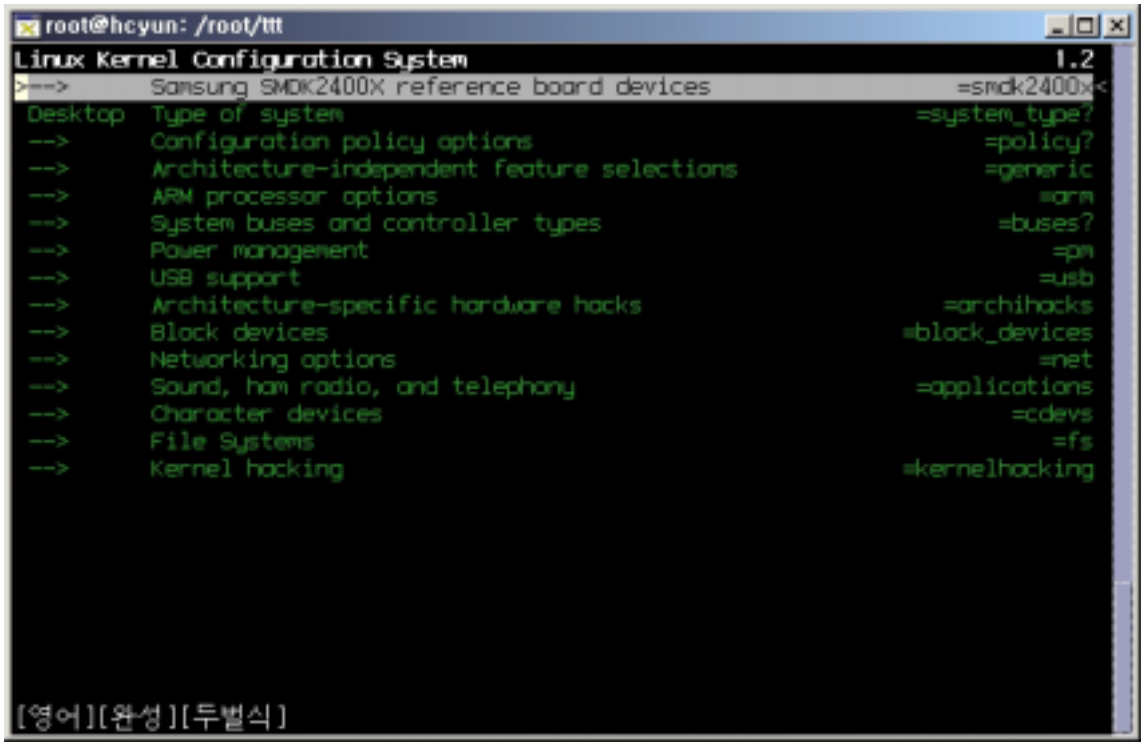


그림 21. Kenel configuration with menuconfig

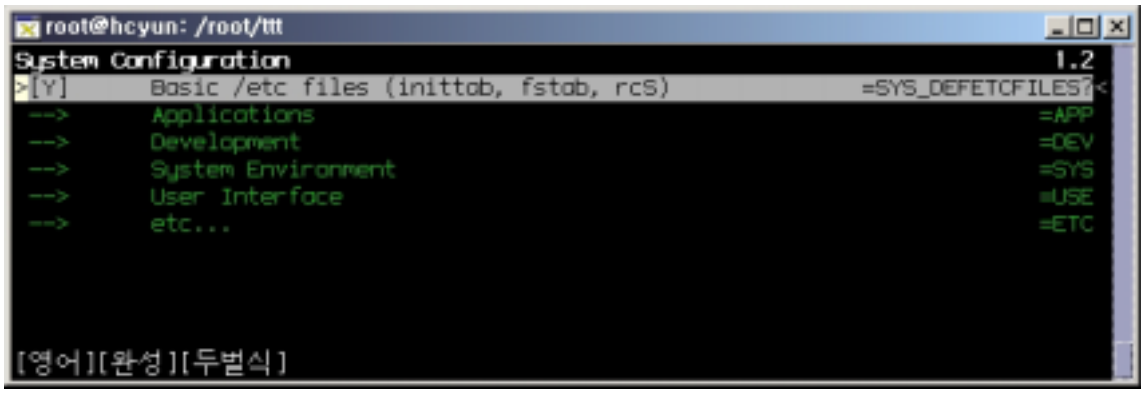


그림 22. System Configuration with Menuconfig

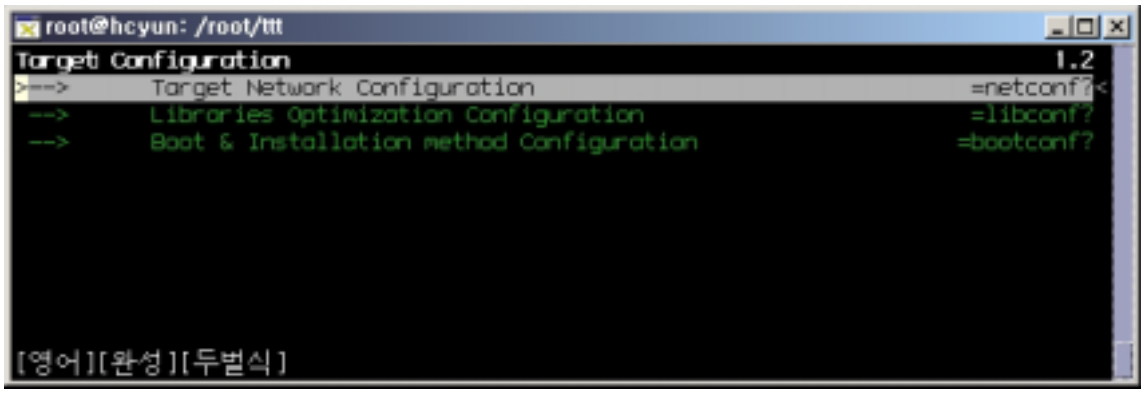


그림 23. Target Configuration with Menuconfig

9.2.2. xconfig

menuconfig 과 마찬가지로 커널의 기본 설정 프로그램이다. 역시 모든 설정항목은 동일하다.

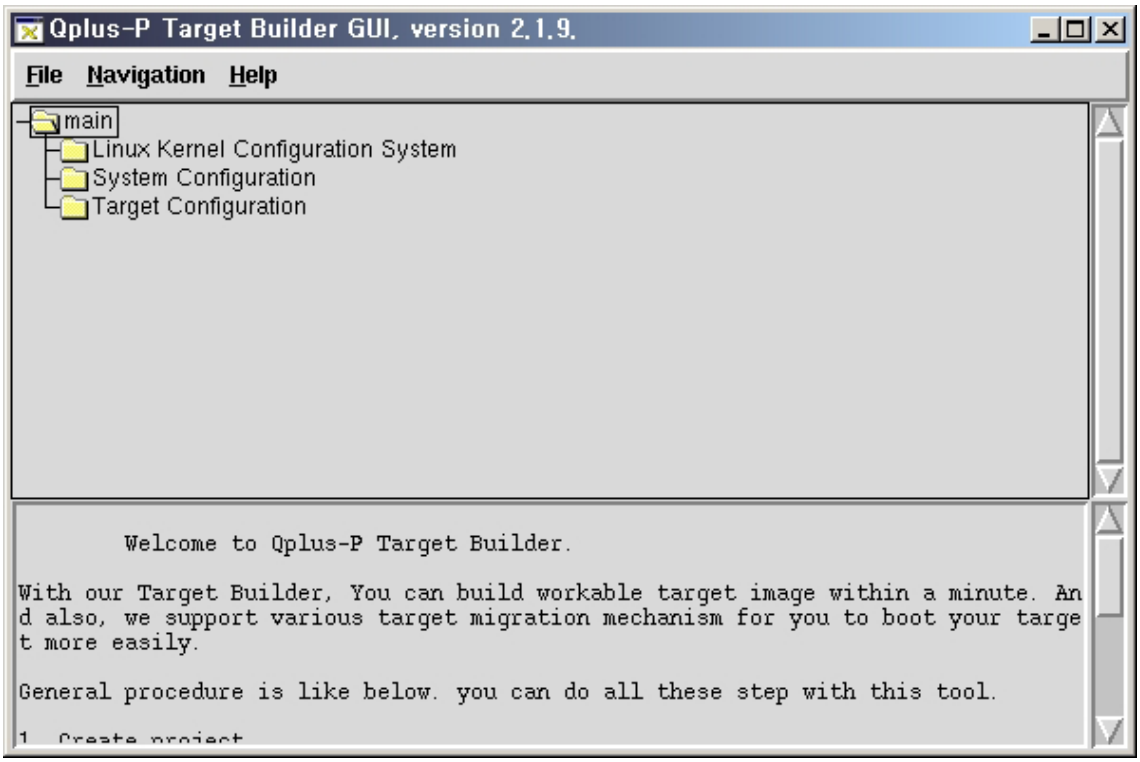


그림 24. Xconfig

9.3. 타겟 이미지 생성

커널과 타겟의 root 파일시스템의 이미지를 생성하기 위해서는 다음과 같이 한다.

```
# make
```

이것은 타겟 빌더 GUI 에서의 Build > Build All 과 동일하다. 따라서 모든 것을 소스부터 컴파일하는 것이 아니며 타겟 빌더 GUI 를 사용할때와 마찬가지로 이전 Build 부터 설정이 변경된 부분만 컴파일 하므로 일반적으로 시간이 오래 걸리지 않는다. 그러나 필요한 경우 다음과 같이 커널과 루트파일시스템을 각각 생성할수도 있다. Make 만 치면 실제로는 make kernel 과 make rootfs 를 연달아 수행하는 것이다.

```
# make kernel
# make rootfs
```

make 또는 make kernel 및 make rootfs 로 생성된 이미지는 다음과 같다.

```
<projdir>/target/kernel/qplus
<projdir>/target/rootfs.tar.gz
```

이렇게 생성된 이미지는 다음장에 기술된 방법에 의해 타겟에 적재된다.

보다 더 세세한 작업을 원하는 사용자라면 kernel.py, buildpkgs.py, mkrootfs.py 프로그램을 사용할 수 있다. 각각의 사용법은 '-h' 옵션을 입력하면 나타난다.

9.4. 타겟에의 적재

타겟에의 적재는 다음과 같이 한다.

```
# make install
```

이는 실제로는 <projdir>/tools/targetinstall 이라는 쉘 스크립트를 수행하는 것이다. 따라서 사용자가 BSP 에 포함되어 제공되는 것 이외의 일을 하기를 원한다면 이 쉘 스크립트를 수정하면 된다.

성공적으로 수행되면 deploy 방법별로 각각 다음과 같은 타겟 부팅방법에 대한 설명이 나온다. 각 항목을 읽고 그대로 따라하면 된다.

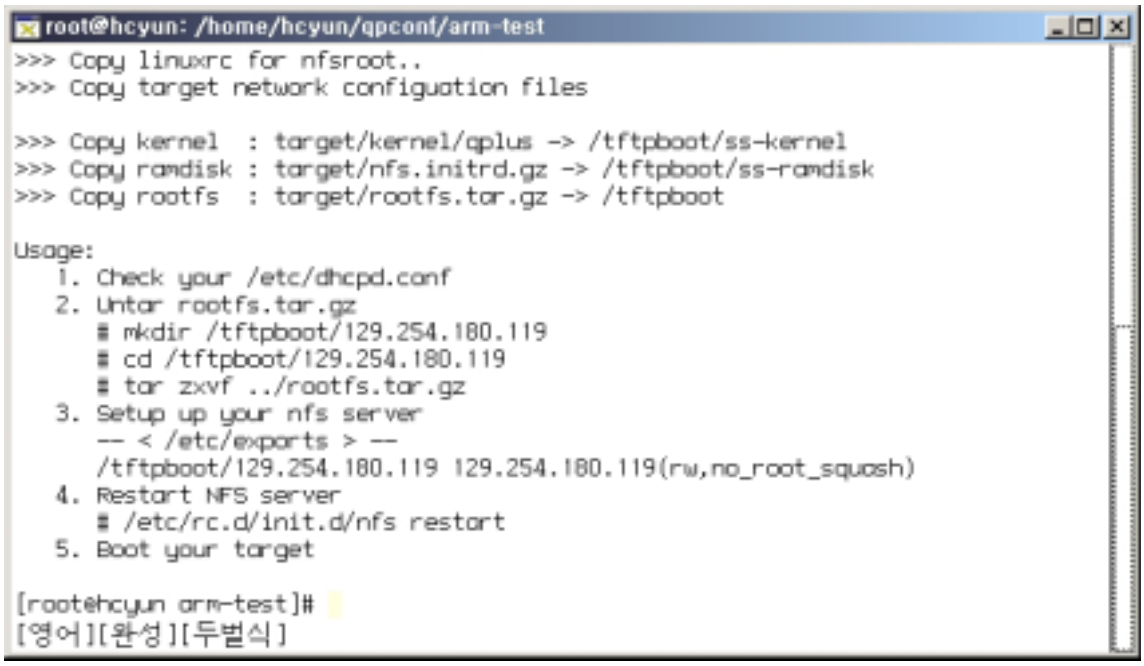


그림 25. nfs deploy instruction