Qplus-P Target Builder User's Guide

Version 1.1

2002/10/1

Woochul Kang(wchkang@etri.re.kr)

Heechul Yun(hcyun@etri.re.kr)

Contents

When target system boots using etherboot disk, it displays MAC address of the network card. You have to

About This Document

Qplus-P is ETRI's embedded Linux solution for internet appliances such as PDA, Digital TV setopbox and webpad. Target Builder is a development toolkit for Qplus-P.

This document describes embedded system development procedures using Qplus-P/Target Builder.

1. Introduction to Target Builder

Target Builder is an embedded Linux development toolkit tightly coupled with ETRI Qplus-P . It provides many features for developers to build embedded Linux system. These features include configuration, dependency checking, conflict resolution, project management and deployment support to the target system. Using Target Builder, developers can make fully functional operating system very easily and quickly.

Main features of Target Builder are as fellow.

- User friendly GUI interface
- CML2-based Integrated configuration system (kernel, applications and system environment)
- Automatic dependency checking and Conflict resolution
- Library optimization
- Fine-grain control of system; file-list, compile option, and more
- Various deployment methods support
- Foot-print reporting
- On-line help
- Project & configuration management

This chapter describes brief introduction and usage of Target Builder.

1.1. Installation

This section describes (un)installation procedures of Qplus-P/Target Builder.

System Requirement

- Linux distribution of development host :
  - RedHat 7.0, 7.1, 7.2
  - WowLinux 7.1 (Korean)

- Other distribution also may work but not tested.

- Required Host Services
  - DHCP
  - TFTP
  - NFS

Installation procedure

You should have following list of files via CD or downloading from our ftp site. For qp-bsp-* and qp-devels-* packages, you may want to download only needed files that match with your target board. Currently 4 BSP in x86 and arm architecture are provided.

```
README                                          <-- this file
install.sh                                      <-- install script
qp-tb-2.0-280902.i386.rpm                       <-- Target Builder
qp-packages-1.0-280902.i386.rpm                 <-- package QPD and srpm
qp-bsp-arm-Zaurus-1.0-280902.i386.rpm           <-- BSP for Zaurus
qp-bsp-arm-iPAQ-1.0-280902.i386.rpm             <-- BSP for iPAQ
qp-bsp-arm-s3c2400-1.0-280902.i386.rpm          <-- BSP for SMDK2400
qp-bsp-i386-generic-1.0-280902.i386.rpm         <-- BSP for generic x86
qp-bsp-i386-etri-hestia2-1.0-280902.i386.rpm    <-- BSP for ETRI hestia2
qp-tools.tar                                    <-- python2.2, ...
qp-devels-arm.tar                               <-- header and doc for arm
qp-devels-i386.tar                              <-- header and doc for i386
```

Actual installation procedure are performed via install.sh script as you can see in Figure1

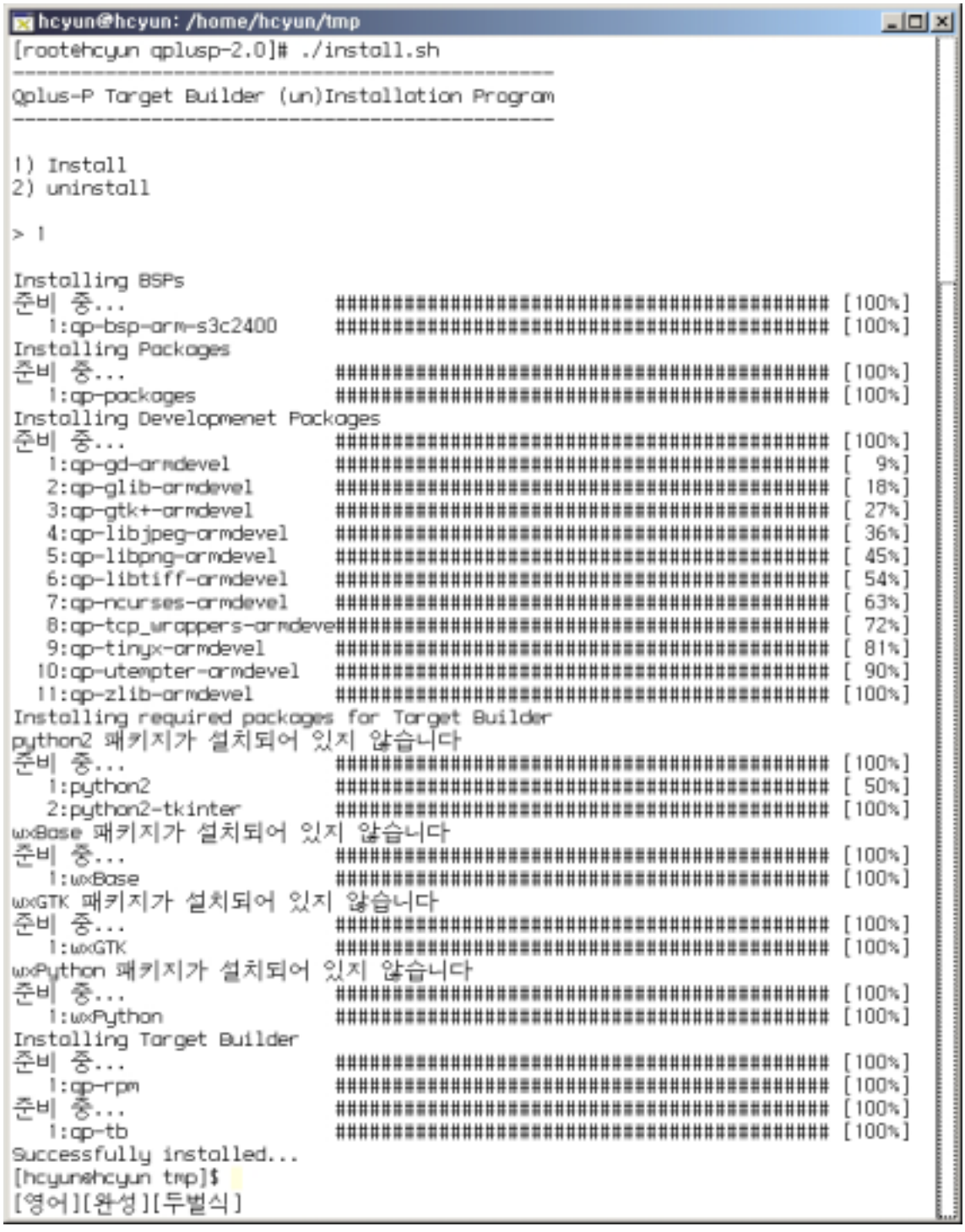


Figure 1. Installation

NOTICE: If your host system has python 2.1 installed, it's recommend to run install.sh after uninstalling python 2.1. Target Builder needs python 2.2

Directory structure after installation

Directory structure after installation should be like Figure 2.

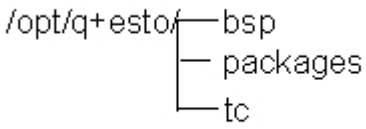


Figure 2. Directory structure

'bsp' directory contains architecture and board specific kernel, deployment tools and precompiled packages.

'packages' directory contains source rpm files and QPD (Qplus Package Descriptor) files.

'tc' directory contains main Target Builder executables and common support programs.

1.2. Running Target Builder

Changing to 'root' privilege

To run Target Builder root permission is required. If you did not login as root, login as root using 'su' command.

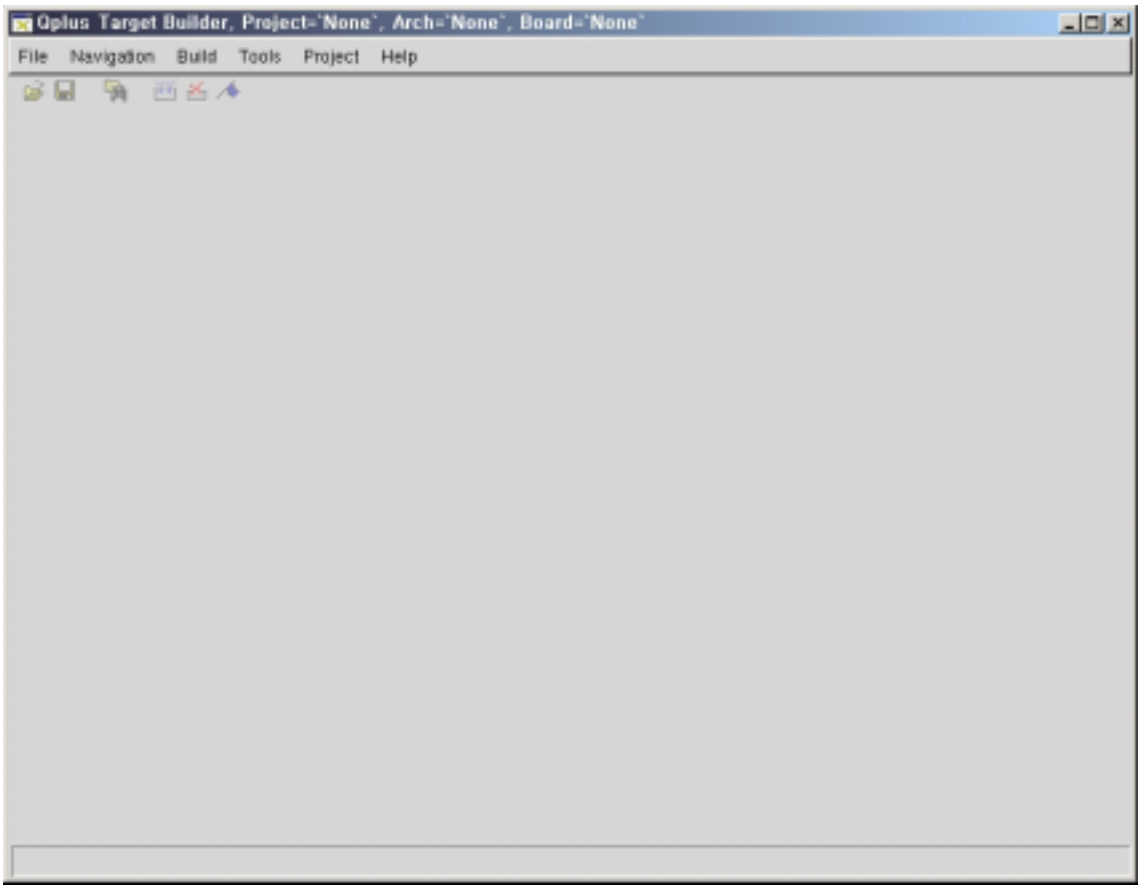
```
# su -
```

Running Target Builder

```
# tb (or /opt/q+esto/tc/bin/tb)
```

1.3. Creating a new project

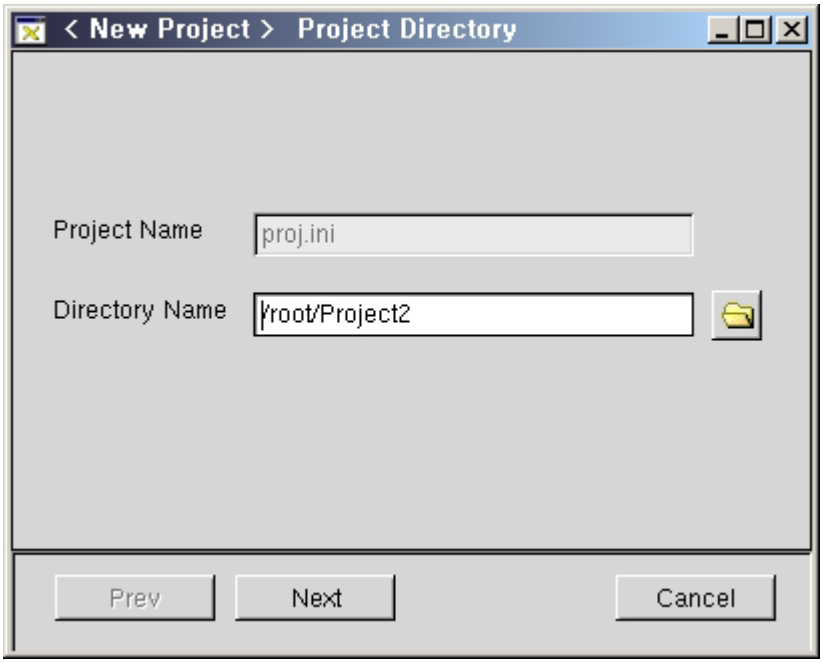
The first time you launch Target Builder, the following screen displays.



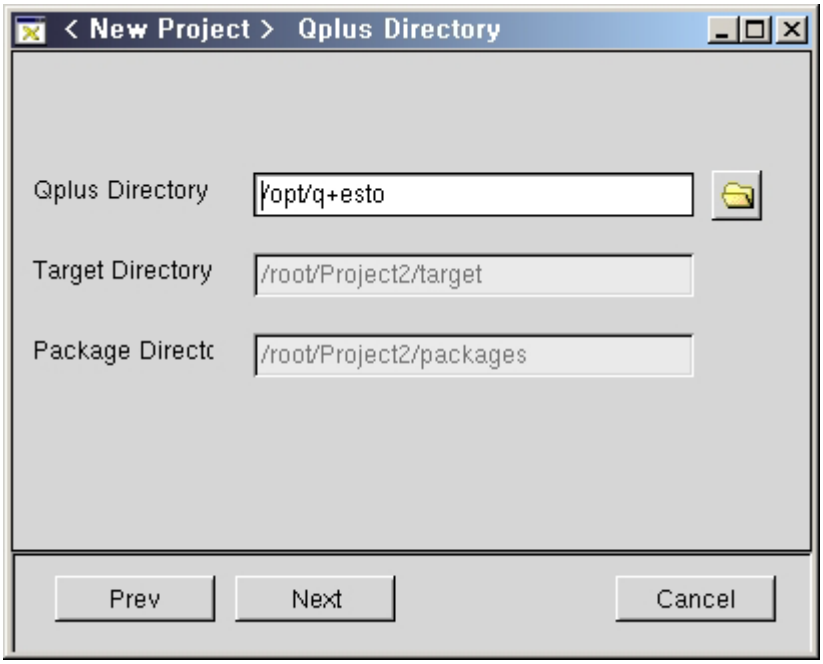
Each project covers creating entire system for each board. To create a new project you must select proper architecture and specific board name for your target. Do following in the menu.

```
Project > New
```

In the following windows, you should specify project directory. After then, press ‘Next’ button.



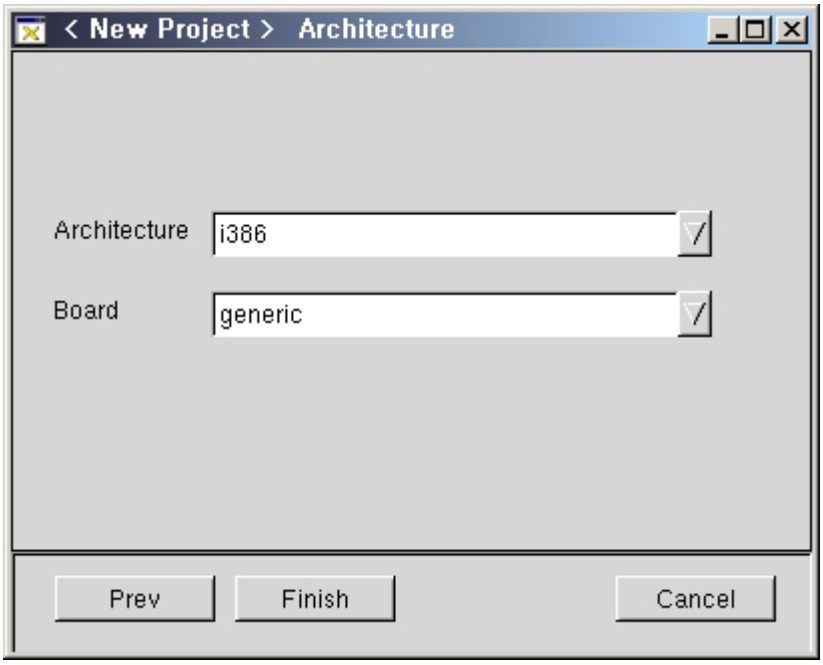
In the following window, specify Qplus-P install directory. Currently Qplus-P install directory is fixed at /opt/q+esto , so just press ‘Next’ button.



Qplus Directory: Qplus/Target Builder install directory.

Target Directory: Generated target system image (kernel, root filesystem) is stored here. In other words, you can see everything that will be loaded on your target.
Package Directory: source rpm and its QPD descriptor for all packages provided by Qplus-P.

You must select target architecture and specific board name of your target.



Architecture: Select target architecture. Currently only *arm* and *i386* are provided
Board: Select specific board name of your target.

List of items in this window may be different as your BSP installation.
If you found proper item, press 'Finish' button. Then Target Builder will create project directory for your target.

Notice : **It can takes up to several minutes. Please wait for a while.**

1.4. User Interface of Target Builder

Figure 3 is initial window after project is loaded.

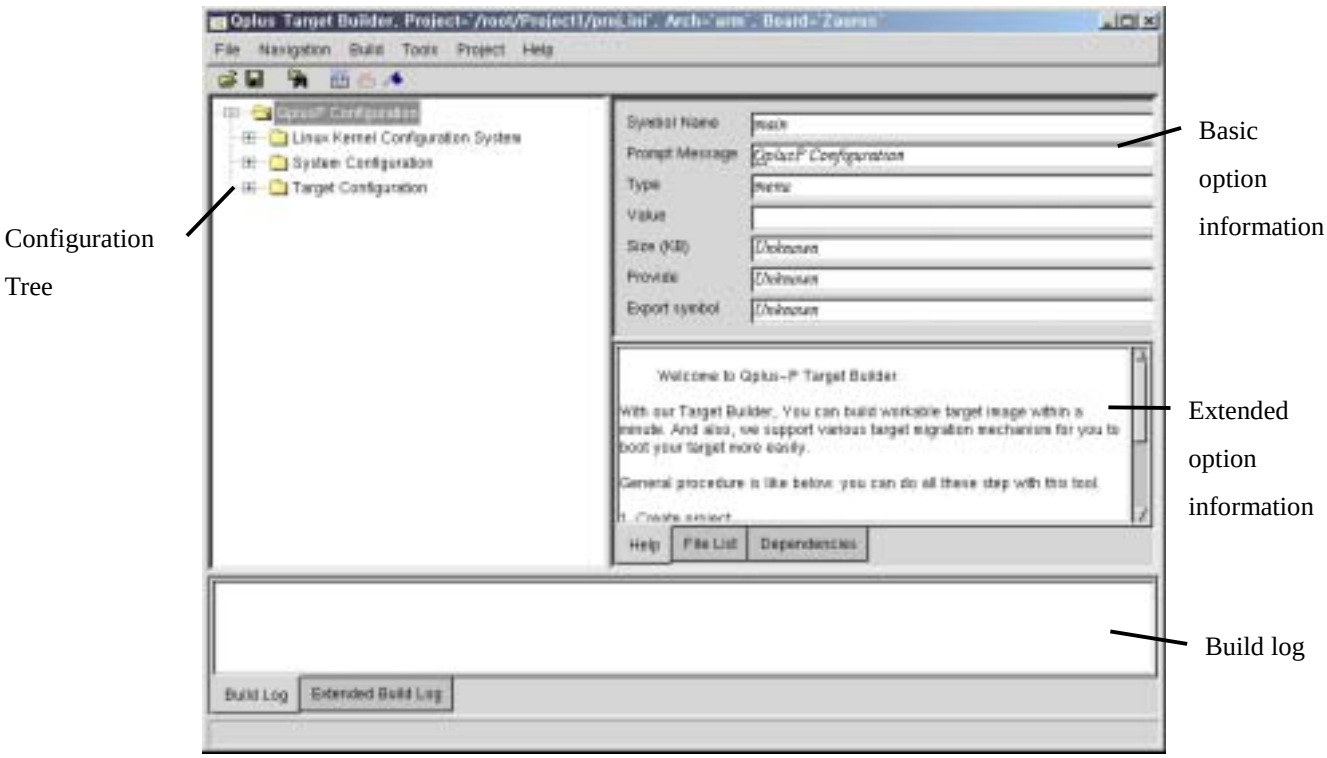


Figure 3. Project loaded window

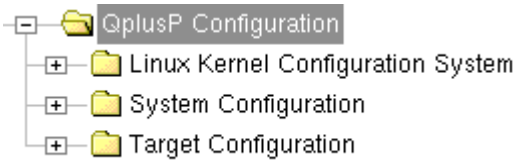
Developers can navigate and click the configuration tree. Various property information are shown in right side of the window. When (s)he start to build the system, build log will be displayed in the bottom box.

1.4.1. Configuration Tree

In Target Builder all configurable options are provided in single configuration tree. That includes kernel, applications, board specific options, etc.

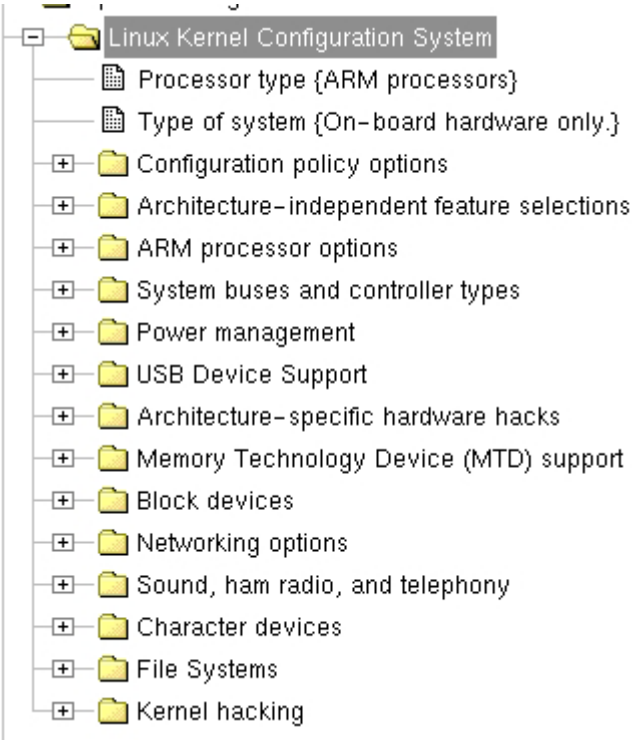
Developer can configure his/her target system through navigating this tree.

Tree structure



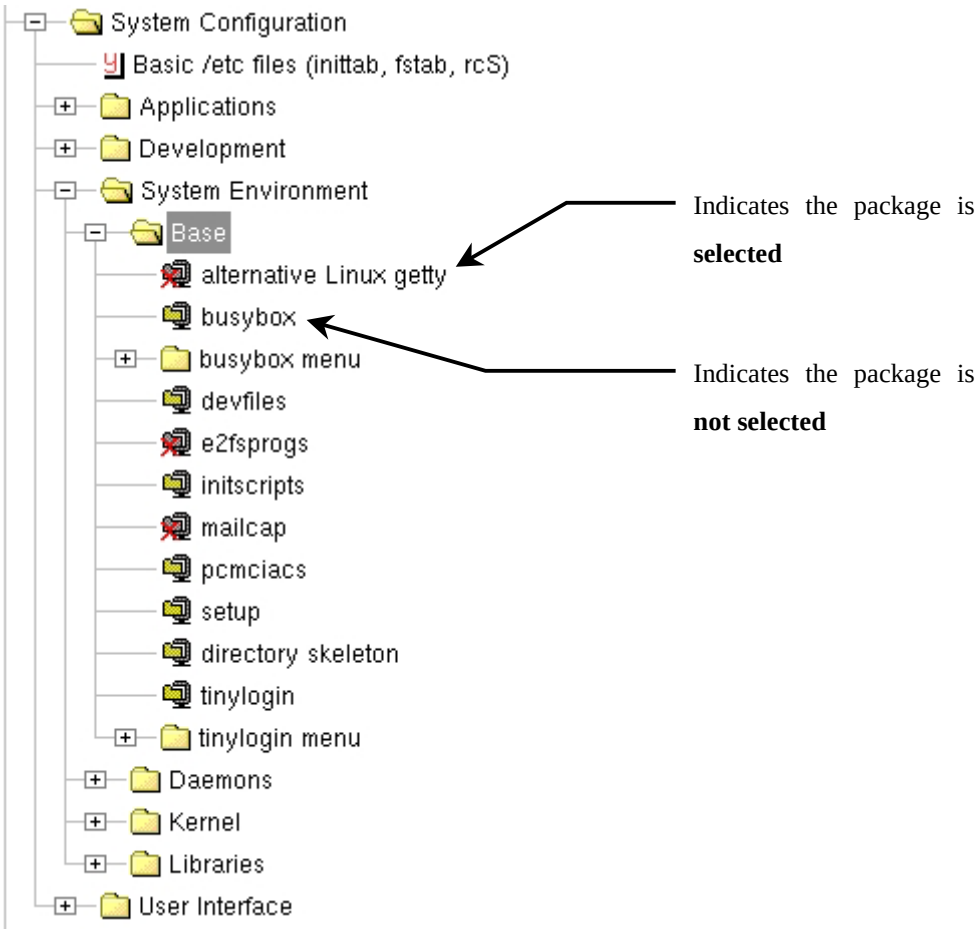
Configuration tree three major parts: kernel configuration, system configuration, and target. Configuration. Kernel configuration part has similar structure with normal Linux configuration and system part has options regarding to application packages. Target part is for target specific options such as network address and booting method. Each part will be explained in following sections.

1. Linux Kernel Configuration System



Linux kernel configuration section is based on Eric Raymond's CML2 rules file of standard Linux distribution.

2. System Configuration



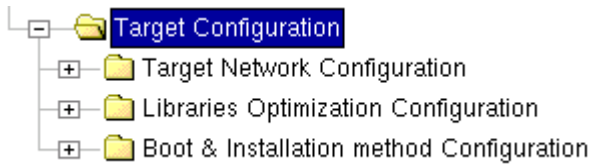
You can configure entire system applications and libraries provided by Qplus-P in the 'System Configuration' section. Each package is categorized based on its group.

Icons of each package shows whether it is selected or not. means it's selected and means it's not selected. Each package can have sub configurations. For example, 'busybox menu' is submenu of 'busybox' package. Unlike packages (deb or rpm) in normal Linux distribution such as Redhat or Debian, package in Qplus-p can be configured more finely through the submenu.

Notice::

Submenu of each package is only displayed when the package is selected.

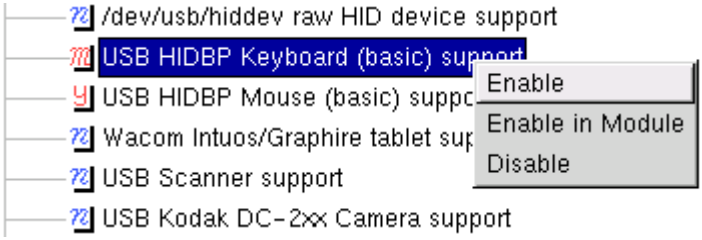
3. Target Configuration



Target Configuration deals with target specific information such as network address and deployment method.

Set value of each option

Point an item that you want to configure and press right button of your mouse. Then you can see a popup menu. Contents of popup menu will be different with the value type of the item.



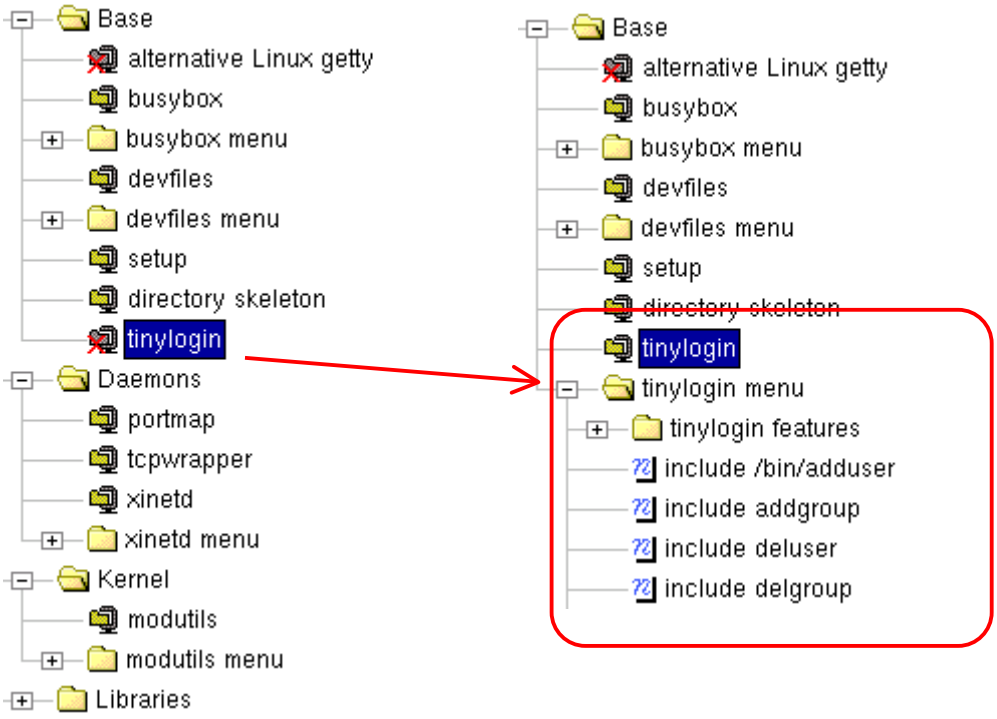
In the case of BOOL type, you can select **"Enable" or "Disable"** . For TRIT type(modulable kernel option), you also can select **"Enable in Module"**. For other types (CHOICE, STRING, DECIMAL, HEXA), a dialog box will be launched to set proper values of each type.

You can see the type and value of each item through icon.

- : BOOL or TRIT type and **Enabled**.
- : BOOL or TRIT type and **Disabled**
- : TRIT type and **Enabled in Module**
- Broadcast address {129.254.180.255} : Other types (Not bool or trit type). Values is displayed inside '{ }'

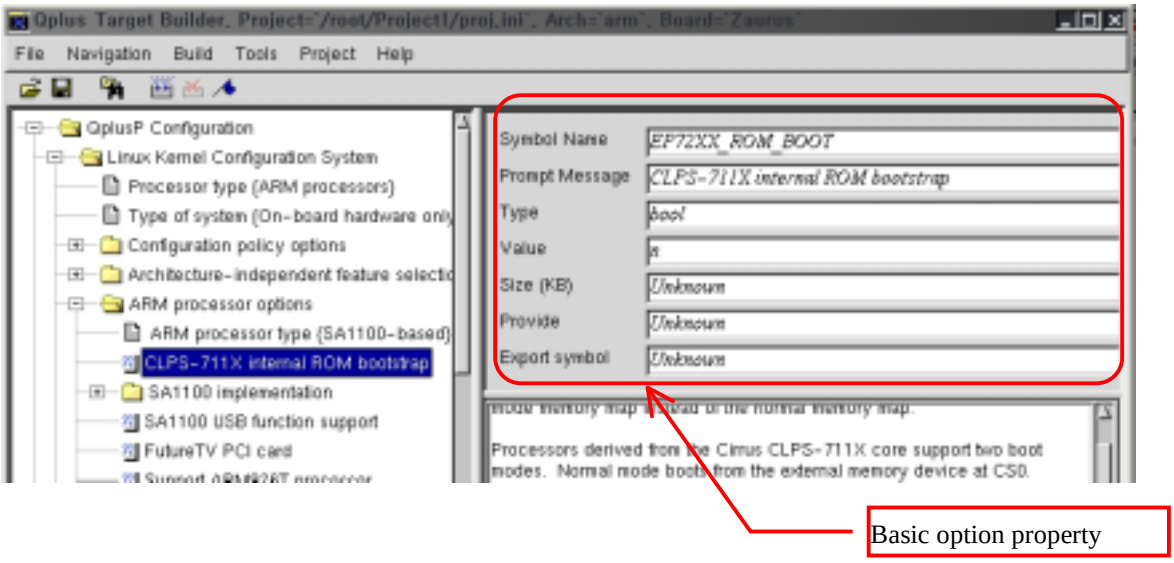
Incremental disclosure of hidden items

Target Builder shows options incrementally. It means that you can start configuration from small set of options and when you enable an option related options, Target Builder show related options that you just enabled.



In the above picture, submenus of tinylogin package is only shown when the package is enabled.

1.4.2. Basic Option Properties



It shows basic properties of selected option. Displayed properties are as follows

Symbol name: CML2 symbol name of the option

Prompt Message: Simple description

Type: Type of the option. There are B*OOL, TRIT, CHOICE, STRING, DECIMAL and HEXA* types.

Value: Value of the option

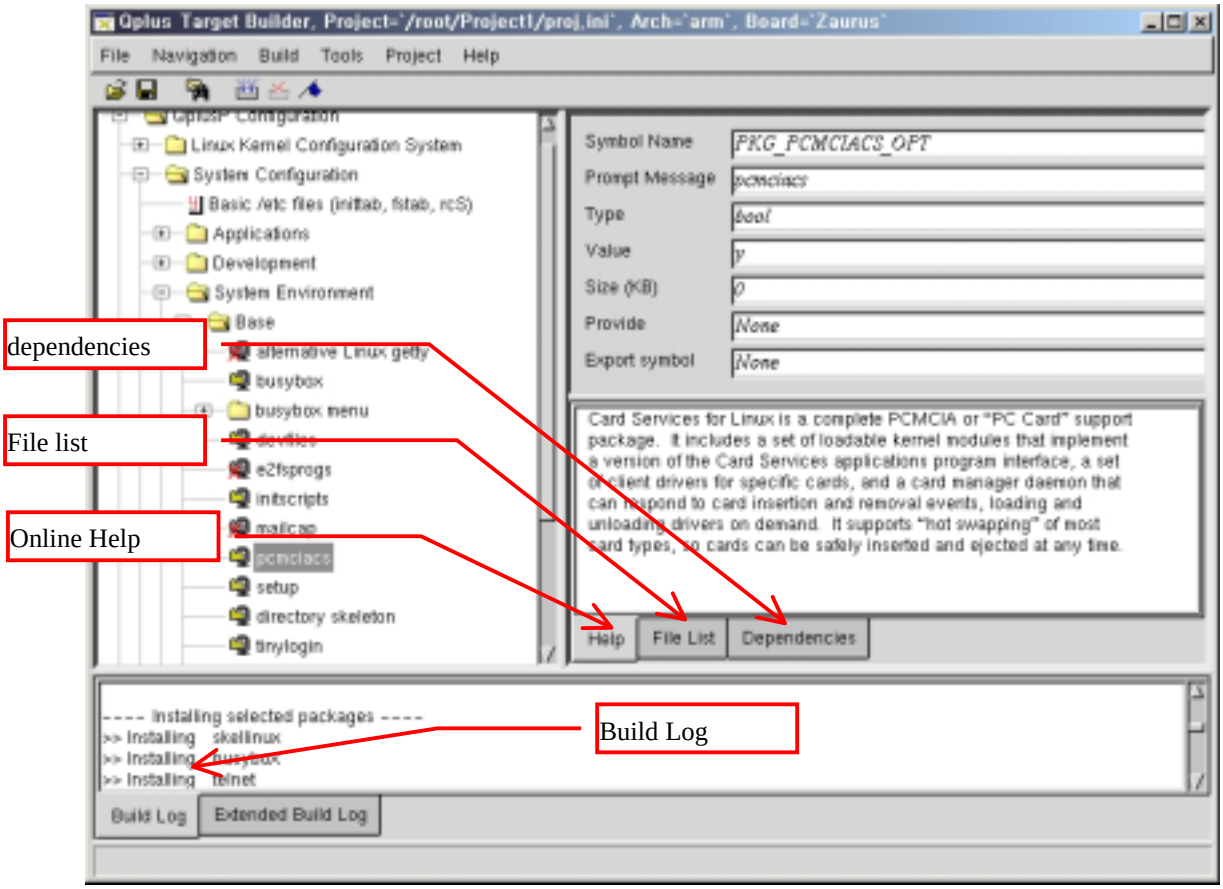
Size: Size of a package including sub options. Only meaningful for package option.

Provide: provided symbol of this item. It is used to prevent conflict.

Export symbol: exported symbol name, which will be stored as a file. (It's special for a busybox and a tinylogin package).

1.4.3. Extended Option Properties

Extended option properties let you know about more detailed description, list of files to be installed and dependency of each option. And '

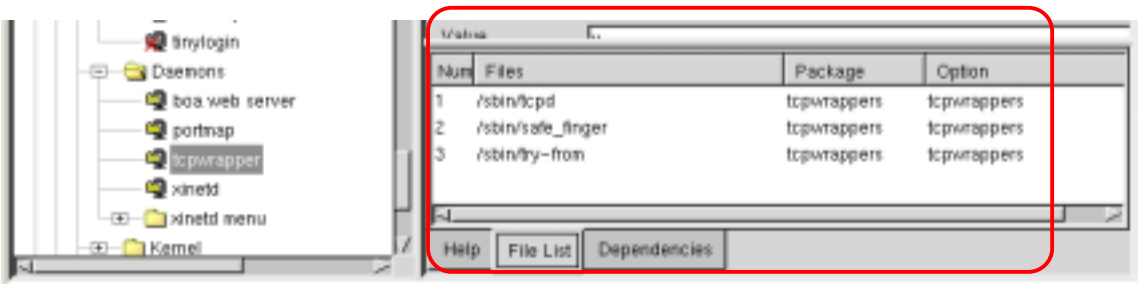


Help Tab

This tab shows detailed description of the selected item.

File List Tab

File list to be installed. It includes files in selected sub items. For example, the following picture shows that *tcpwrapper* package install 3 files *(/sbin/tcpd, /sbin/safe_finger, /sbin/try-from*) to the target system.



If you select top-level item *Qplus-P Configuration* you can see the whole file list of your target root filesystem.

Dependencies Tab

This tab shows dependency information of each item. Refer section 2.7.

1.5. Save and Load

1.5.1. Save

Each project has its configuration file (config.out) in the project directory. Target Builder stores whole configuration in the file and when you open a project Target Builder will load last configuration status from the file, *config.out*

Notice: Configuration states are automatically stored when you close project or start to build. Also you can save it anytime using save menu, ***File > Save***.

1.5.2. Save as and Load

You can save and load configuration state at arbitrary file.

Save As

File > Save As : stores current configuration information on the given file name.

Load

File > Load : loads configuration information from a selected file.

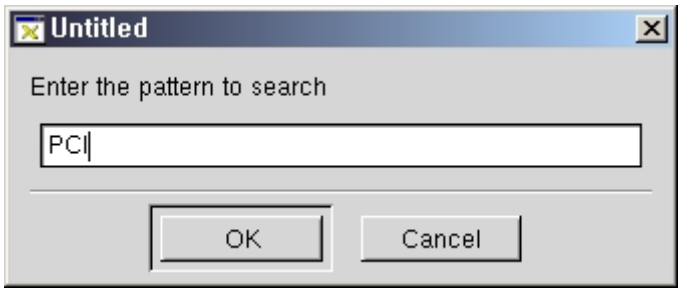
1.6. Search

Target Builder provide powerful search feature. You can search items, which contain a given word in help text, symbol name, or prompt message.

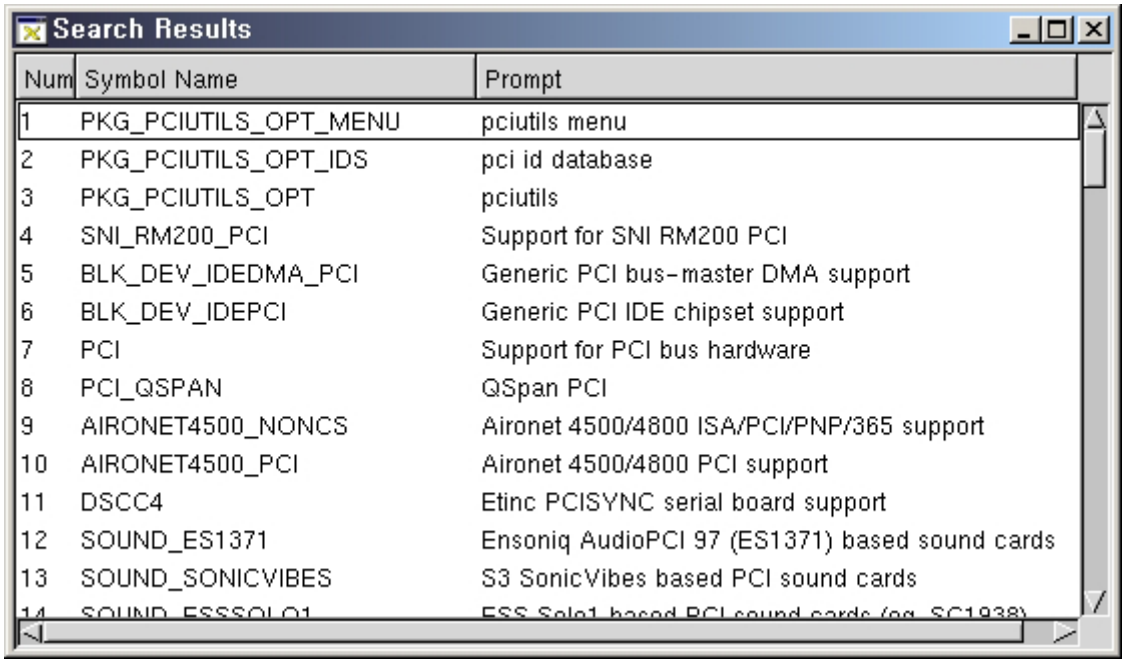
1.6.1. Search Symbols (Alt+F8)

This feature enables you to find items that contain given string in the prompt message or symbol name. For example you can find all item that contain 'PCI' in the symbol name or prompt message.

Select ***Navigation > Search symbols*** in the menu



type 'PCI' in the message box.



Search result window will be created. If you double click an item, point will be located to the item in the configuration tree.

1.6.2. Search help

Same as *Search Symbols* except it tries to find a given string in the help text.

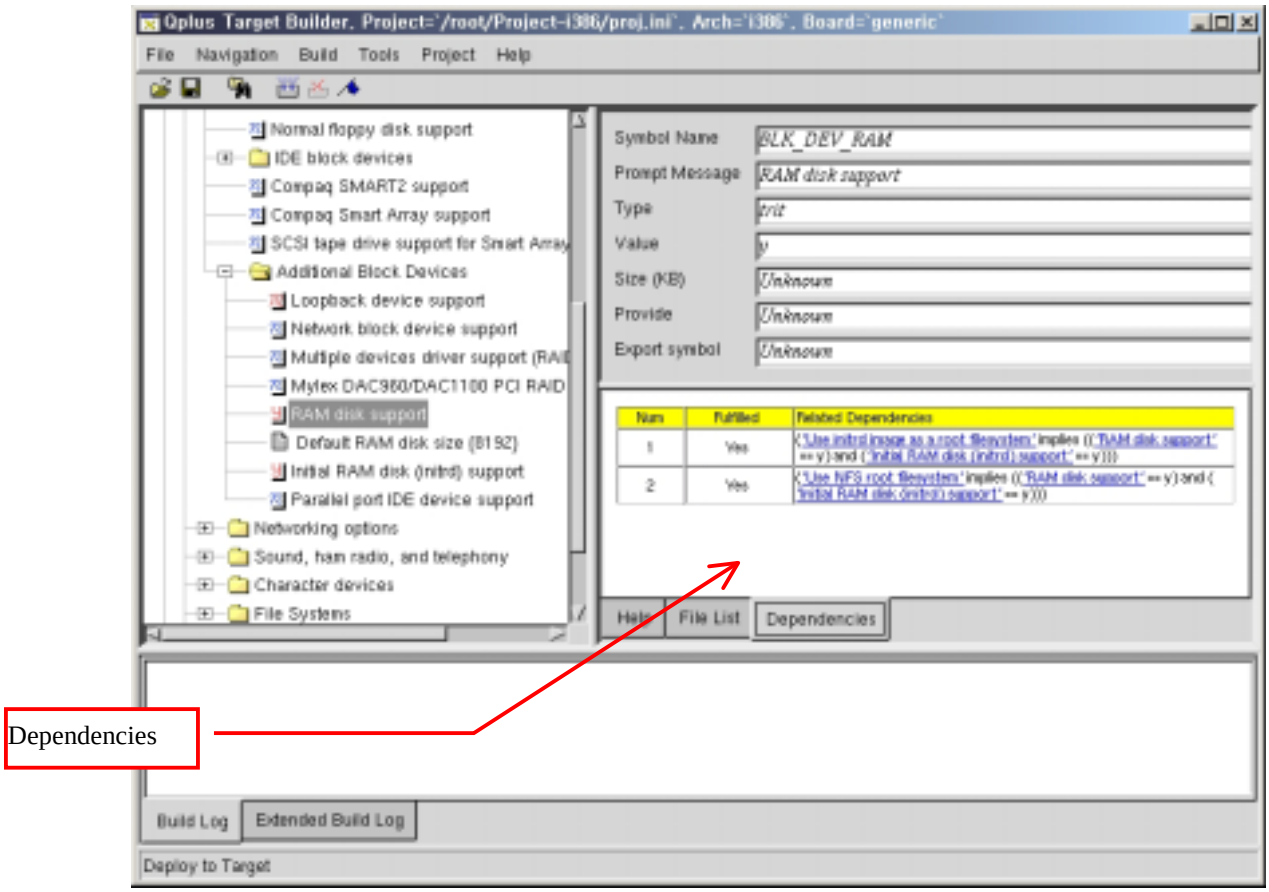
1.6.3. Goto

You can use this feature when you know exact name of symbol and you want to know where the symbol is in the configuration tree.

1.7. Dependency

Many configuration items have dependencies. For example, X-window or syslog daemon requires *'unix domain socket'* feature of kernel configuration to be enabled. This kind of dependencies can be described through QPD or CML2 language.

Target Builder shows dependencies of each item through the *'Dependencies'* tab and you can go-to dependent item directly by clicking the mouse.



1.7.1. Logical expression of dependencies

Dependencies are shown in logical expression form and describe the predicate of safe state. Logical operators that are used in the expression are like below.

and

or

not:

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

Implies (means ⊃ or →)

For a real example, If you select ‘Unix Domain Socket’ item in the configuration tree you can see following 2 dependencies in the dependencies tab.

.

(‘include syslogd’ implies (’unix domain socket’ == y))

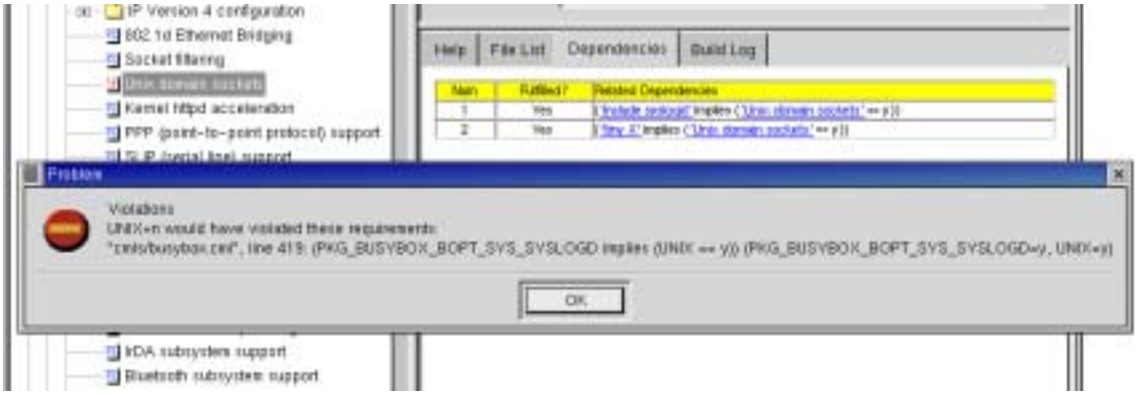
(‘tinyx’ implies (‘unix domain socket’ == y))

The first line means, enabling ‘include syslogd’ option implies ‘unix domain socket’ should be true. In other word, ‘unix domain socket’ option is needed to be true to enable ‘include syslogd’ option. The second line is similar with the first line. Underlined blue is prompt message of each configuration option and you can click it to go to that option to see what this option is.

1.7.2. Dependency violation

Target Builder checks dependency-violation every time you change a value of any option item. In case of violation, it reports what causes the violation.

In the example of previous section, there are dependencies between ‘unix domain socket’ and ‘tinyx’. If you try to disable ‘unix domain socket’ option while ‘tinyx’ is enabled, it violates the dependency rules. Therefore, you can see message box like following figure.



1.8. Build

This section explains about compiling kernel and applications and generation of target root filesystem.

There are following menu items in the build menu.

Build Kernel: compile kernel only when you have made any change in kernel configuration from last time kernel build. *Build > Build Kernel*

Build Kernel –Force: compile kernel anyway. *Build > Build Kernel –Force*

Build Applications: compile packages in system configuration tree. Not every packages are compiles, but only packages which you have changed compile related configuration (compile flag, …). *Build > Build Application*

Build Root filesystem: generate root filesystem in *<projdir>/target/rootfs*. This directory contain fully functional root filesystem image. Depending on your target configuration target specific configuration such as network address and boot scripts are also generated.

Build All: Do 'Build Kernel', 'Build Application', and 'Build Root Filesystem' in sequence. It is probably most user want to do. You can do this from the menu or you can use short-cut <F8> function key.

Build Target deployment image: process root filesystem depending on your deployment method and your BSP. For example, if you choose initial ramdisk deployment method in i386/generic BSP, Target Builder will generate Etherboot image, which include kernel and ramdisk image of your target root filesystem. Refer to BSP documentations of your board.

Stop: stop the current build. *Build > Stop*

Warning : clicking '*Stop*' menu doesn't mean immediate stop. Currently, it can stop at several checkpoints. we recommend wait a while for safe stop.

2. Target Image Generation

In the previous chapter, we described brief introduction of Target Builder. In this chapter, we describe practical usage to create kernel and target root filesystem.

2.1. Kernel and Package Configuration

When you create a new project Target Builder will load a default configuration of the selected BSP. The default configuration is tuned to fit for most users. Therefore you may don't need to modify anything except target specific configuration such as network address. Later you can configure the target from the default configuration to optimize for your target system. Therefore, in this section we only discuss target specific configuration. For kernel and each package configuration, please refer to online help.

2.2. Network Configuration

This is generic TCP/IP network configuration. For now only static-IP configuration of single Ethernet interface is supported. But you can easily extend to support various type of network configuration such as dhcpd or multiple adapter configurations. Figure 1 shows network configuration menu in the Target Builder. You can configure this by doing as follow.

- Enable *'Target Configuration > Using Static IP'*
- Set proper values of *'Static IP Configuration submenu'*
  - 'Eth0 device name' section is only used when you select your Ethernet device driver as a module. If you included the driver in the kernel, you can leave this field empty.

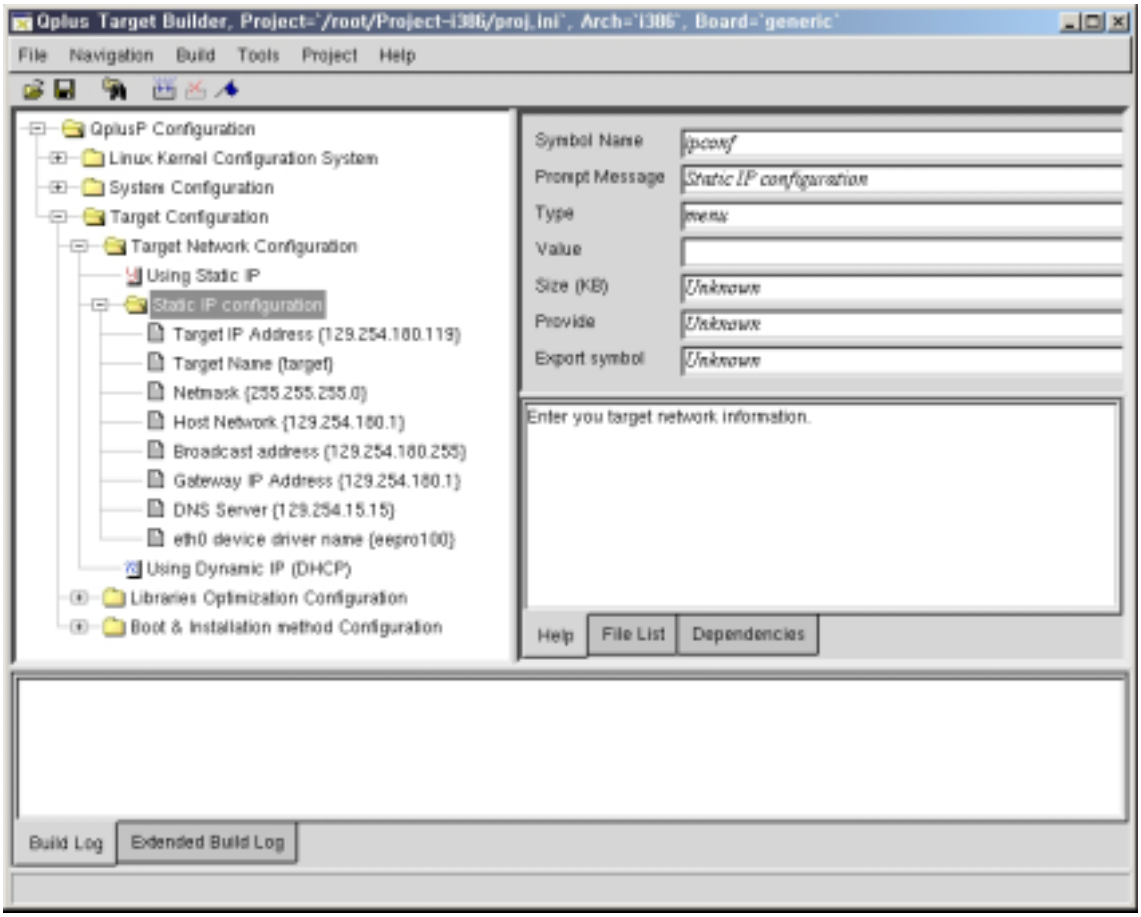


Figure 1. Network Configuration

2.3. Library Optimization

Target Builder provides following two methods of library optimization.

File level optimization

If you enable this option, While generating target system image, Target Builder inspects all executable files and remove unnecessary shared libraries to execute them. This is simple and most useful for most of system.

Symbol level Optimization

If you enable this option, Target Builder further optimizes share libraries in symbol level. It means that only needed symbols are included for each shared library. However this takes a long time and only glibc library can be optimized for now.

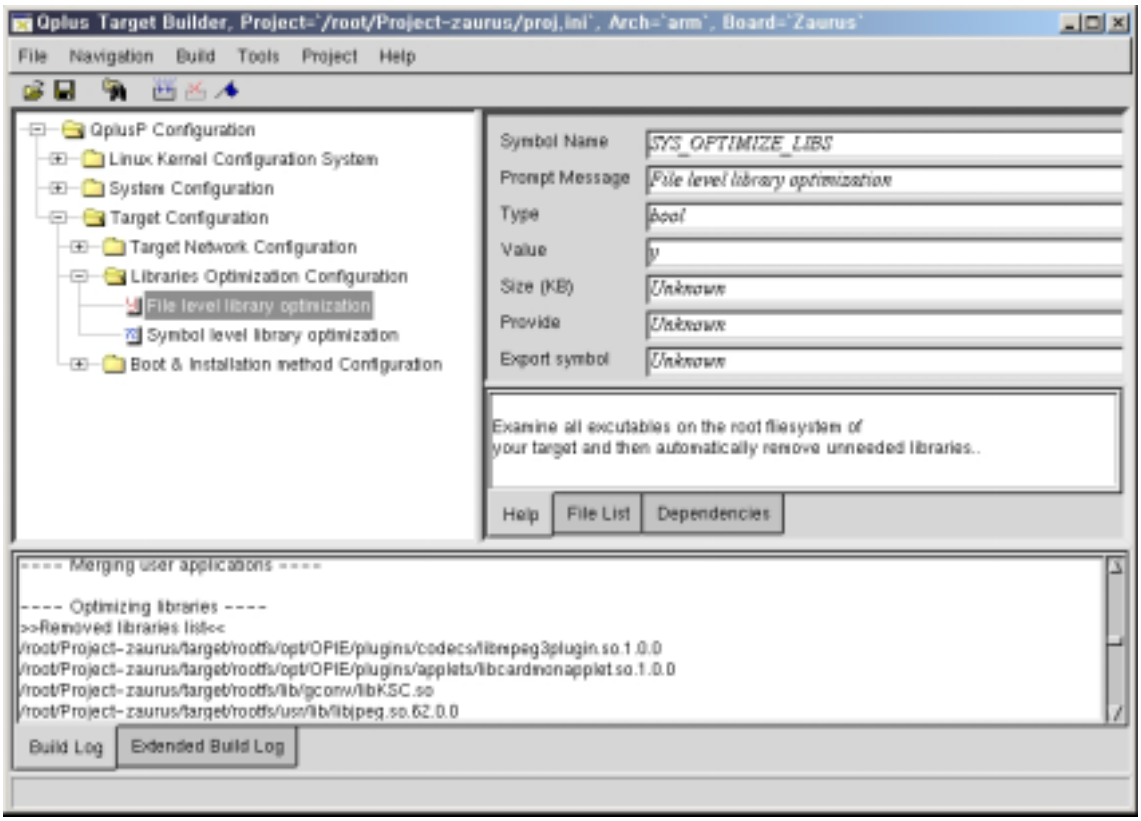


Figure 2. Library optimization

For most case, file level optimization will be enough. If size is really matter you can use symbol level optimization. However, if you enabled symbol level optimization, whenever you add new executables on your root filesystem, you must regenerate whole root filesystem using Target Builder.

Warning: Symbol level optimization have bug for now. It will be fixed in a near future.

If you want to leave specific library files untouched, write down those library names in *<projdir>/piclib/keeplist* file.

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

Example of *keeplist* file

2.4. Build

In this section, we will describe how to build kernel and applications.

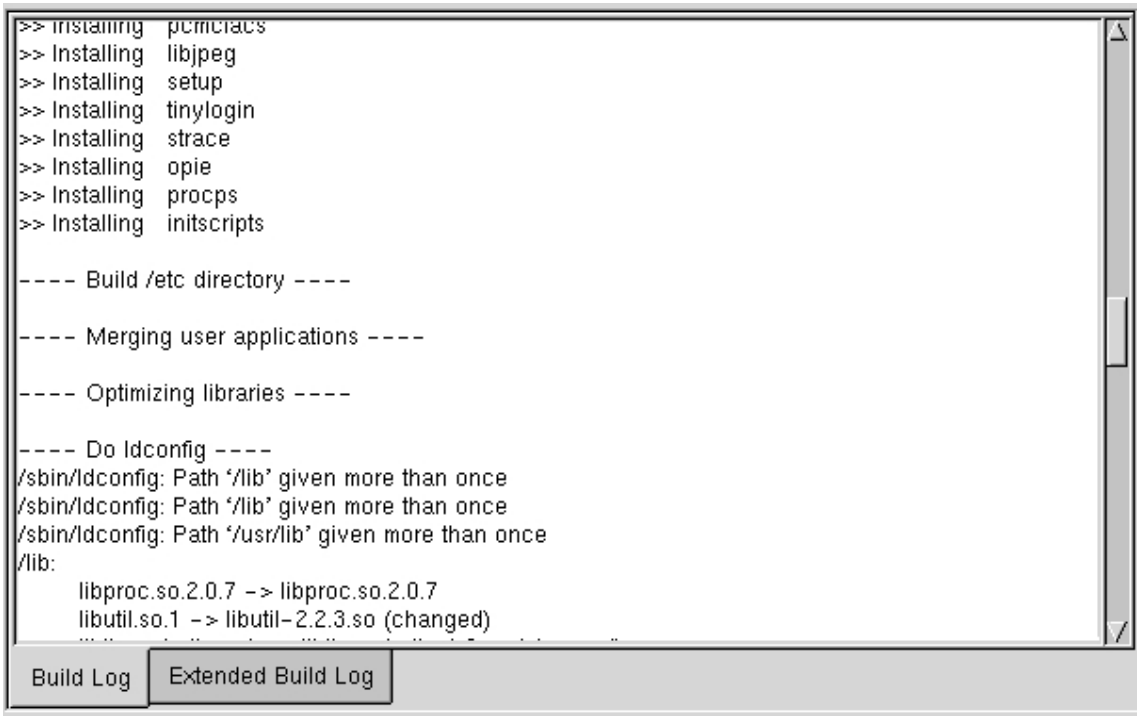
2.4.1. Build All

Select following menu (shortcut is F8)

Build > Build All

This procedure build everything and generate fully functional root filesystem image. In fact, this procedure calls *Build Kernel*, *Build Applications*, and *Build Root Filesystem* in sequence. Each procedure will be explained in the following sections. This doesn't recompile everything but compile only selected components that have changed its configuration from the previous build. Therefore, it doesn't take much time and you may not want to use other build menu if you know exactly what you are doing.

You can see build log in the *Build Log* tab at the bottom of Target Builder.



We will explain other build menu in the following sub sections.

Build Kernel

This will compile kernel and leave kernel image on <projdir>/target/kernel directory and modules on <projdir>/target/usersupp/lib/modules/<kernelver> directory. Because this procedure compares current state with previous build time state, if you didn't changed kernel configuration, it does nothing.

Build Kernel – Force

Same as *Build Kernel* except this doesn't compare state change. Always do kernel compile.

Build Application

Build every selected package in the *System Configuration* section of tree. Like *Build Kernel*, it also compares current state with previous build time state. Therefore a package will be compiled only when the state have changed. Moreover, if you changed a option which does not related with actual compilation such as file list related option, then it doesn't recompile the package and only affected files will be installed (or uninstalled) while generating root filesystem.

Build Root Filesystem

This procedure generate root filesystem directory of target on <projdir>/target/rootfs directory. Detailed procedures are as follows.

- Installing compiled applications (via Build Applications) on the <projdir>/target/rootfs directory
- Generating basic configuration files or boot scripts (e.g. /etc/rc.d/rc.sysinit) based on your configuration.
- Library optimization (if selected)

3. Deployment to Target (i386/Generic)

'Deployment' means to transfer kernel, root file system image which were generated by build process to the designated target system.

In this chapter, we will explain deployment methods in the case of generic PC environment which uses i386-generic BSP.

3.1. Host Requirement

First of all, you needs a target system(i386 based PC) and a host PC runs on Linux(strongly recommends Redhat 7.1 or later). Both target and host system should be connected via Ethernet LAN.

You have to check below functionalities in the host system.

1) Support of loopback device

   Execute following commands with root permission.

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

   If all sequences are executed successfully, it means loopback device is supported.

2) Support of minix filesystem

   Execute following commands with root permission.

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

   if there's a 'minix' item in the result, it means minix filesystem is supported.

3) Installed dhcpd or not

   check /usr/sbin/dhcpd file.

4) Installed tftpd or not

   check /usr/sbin/in.tftpd file.

5) support nfs server or not

check “/etc/init.d/nfs start” command if it executed or not

3.2. Making Etherboot boot floppy disk

Using Etherboot, i386 based target system can download its kernel, root file system from a host system.

‘<projdir>/tools/etherbooimgs/’ directory has etherboot floppy disk images for diverse network card or you can download it from http://rom-o-matic.org. You can find a file with .lzdsk file extension correstonding to your target system’s network card. Then do the following jobs to make a bootable etherboot floppy disk.

1) insert floppy disk.
2) execute, ‘dd if=<device name>.lzdsk of=/dev/fd0’.
3) insert the boot floppy disk to the target system, and boot it.

While booting, etherboot will send bootp request after searching the network card. Refer to section 4.4 for the network configuration of the host system.

When target system boots using etherboot disk, it displays MAC address of the network card. You have to remember it, because it will be used later for dhcpd configuration which will be explained in the next chapter.

3.3. Making Etherboot bootable CD-ROM

‘<projdir>/tools/etherbooimgs/’ directory has etherboot cdrom images for diverse network card or you can download it from http://rom-o-matic.org. You can find a file with .iso file extension correstonding to your target system’s network card. Then do the following jobs to make a bootable etherboot cdrom.

1) make <devicename>.iso file into bootable CD-ROM with CD-Writer.
2) configure first boot method to CD-ROM in the BIOS of target system.
3) insert the boot floppy disk in the target system, and boot the target system.

3.4. Host system configuration for Etherboot

Host system should have proper set up for network daemons needed for etherboot. DHCPD, TFTPD and NFSD are required for etherboot.

3.4.1. Configuration of DHCPD

Configure /etc/dhcpd.conf file as following example. ‘*hardware ethernet’* is a MAC address of target

system , and *'fixed-address'* is an IP address of target. *'filename'* is a name of bootable image of kernel and root filesystem generated by Target Builder.

After the modification, restart dhcpd daemon by typing, *'/etc/init.d/dhcpd restart'*.

```
subnet xxx.xxx.xxx.0 netmask 255.255.255.0 {
host homeserver {
        hardware ethernet xx:xx:xx:xx:xx:xx;
        fixed-address xxx.xxx.xxx.xxx;
        filename qplusp.etherboot
    }
}
```

Example of /etc/dhcpd.conf file

3.4.2. Configuration of TFTP daemon

Modify disable record in *'/etc/xinetd/tftp'* file to *'no'* like below example.

After the modification, restart xinetd daemon by typing, *'/etc/init.d/xinetd restart'*.

```
service tftp
{
        .
        disable = no
        .
}
```

Example of /etc/xinetd.d/tftp

3.4.3. Configuration of NFS server

You should modify *'/etc/exports'* file according to your environment. Following example shows that *'/tftpboot/X.X.X.X'* directory is exported to outside.

After the modification, restart NFS daemon by typing, *'/etc/init.d/nfs restart'*.

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

Example of /etc/exports file.

3.5. Deployment with initrd root filesystem

Now, all setup required for Target Builder was finished. From this section, we will explain how to configure diverse method of deployment using Target Builder.

Initrd is a ramdisk image of target root filesystem. Because ramdisk has size-constraint of 4M~8M, deployment using initrd root filesystem is useful for small target system(in memory size).

The follow steps show how to deploy using initrd

1. Select *'Use Initrd as a root filesystem'* option.

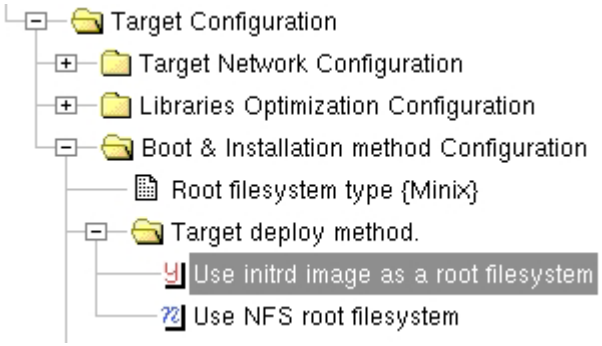


Figure 2. Inird deployment option

2. Select *Build* > *Build All* in the menu.

   Although execute 'Build' in kernel or application configuration, execute 'Build All' in the menu again, because kernel or application program configuration can be changed by deploy option. Following figure shows you dependency rules related to *'Use Initrd as a root filesystem'* option.

   If you select initrd deployment method, *'RAM disk support'* and *'Initial RAM disk (initrd) support'* option in the kernel will be enabled automatically.

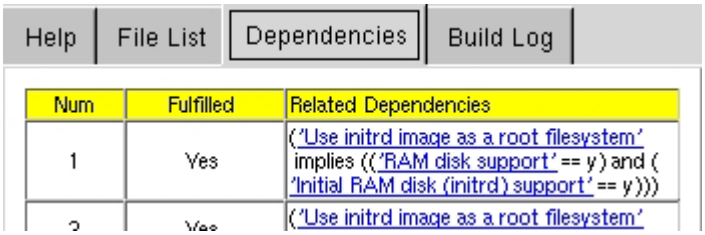


Figure 3. Dependency rules to be check with initrd deployment method

3. Select *Build* > *Deploy Target Image* in the menu

   If it's successful, you can find etherboot image comprising kernel and initrd in *'tftpboo'/qplusp.etherboot'*

Separate kernel and ramdisk image can be found also in *'<projdir>/target/kernel/qplus'* and *'<projdir>/target/rootfs.img'* file respectively.

4. Check DHCPD and TFTPD are working on you host system.
   Refer to section 4.4 for the setup of the services.
5. Boot the target system.
   Check etherboot bootable floppy or CD-ROM is inserted(see the section 4.2 and 4.3). After booting, *'qplusp.etherboot'* file will be downloaded from the host system and conventional linux boot procedure will follow(if your target system has no VGA and keyboard, use serial console. Ensure 'Use serial console' option enabled for this purpose).

3.6. Deployment with NFS

This deployment method uses a particular directory of a host system as a root directory of a target system via network connection. This a very useful method in the development stage, because you can add, delete and modify files in the host system easily.

The following steps show how to deploy using NFS root filesystem.

1. Select *'Use NFS root filesystem'* option, and then select sub-options (nfs server configuration).

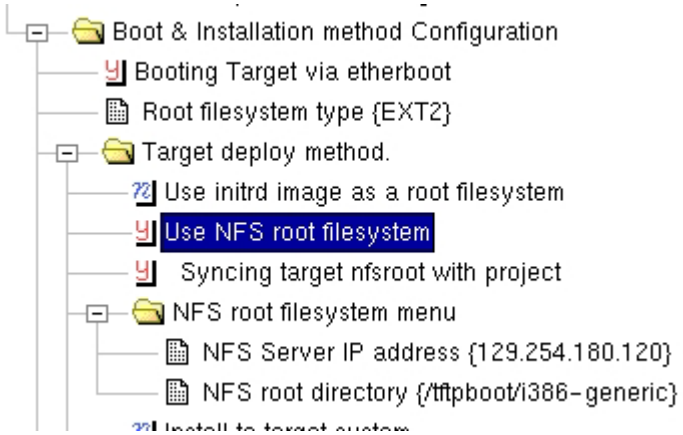


Figure 4. NFS deployment option

If you select *'Use NFS root filesystem'* option, two optons, *'Syncing targetnfsroot with project'* and *'NFS root filesystem'*, will come out. *'Syncing targetnfsroot with project'* option makes Target Builder to synchronize the contents of newly generated target root filesystem(in <projdir>/target/rootfs) and exported NFS directory(including sub directories) using rsync command. If the option is not selected, Target Builder generates only a tar archive file with the generated target root filesystem.

'NFS Server IP address' in the NFS root filesystem menu is for an IP address of NFS server(generally, will be host system running Target Builder). *'NFS root directory'* option is for a directory exported by a NFS server. These options' value is used as basic configuration information about NFS server in booting target system.

2. Execute *Build* > *Build All* in the menu.
3. Execute *Build* > *Deploy Target Image* in the menu.

   Now, 'tftpboot/qplusp.etherboot' file is generated, and target root filesystem is generated also according to the result of the configuration above. If *'Syncing targetnfsroot with project'* option is selected, exported NFS root directory will be updated, otherwise '/tftpboot/rootfs.tar.gz' file will be generated. 'rootfs.tar.gz' file needs to be extracted by hands.

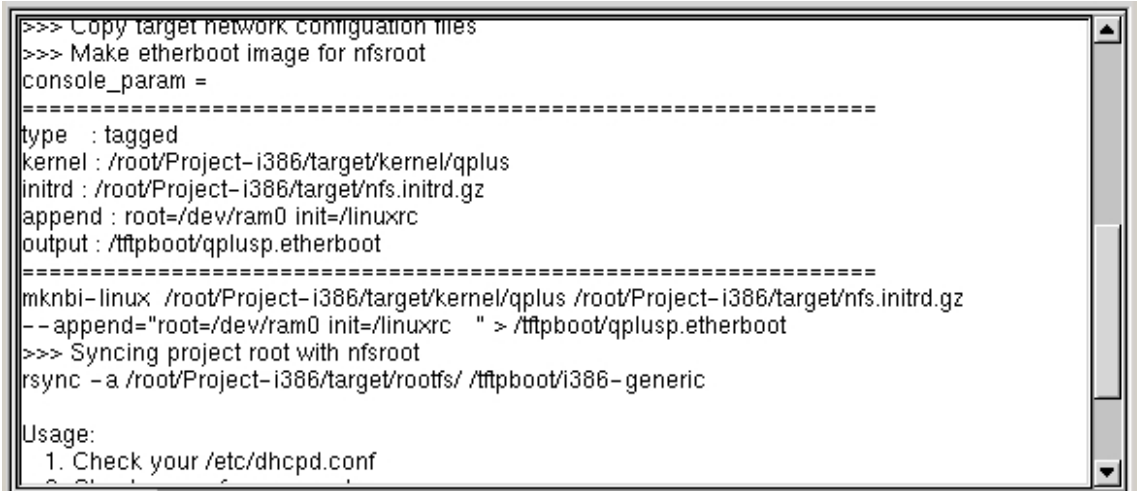


Fig 5. logging display when nfs deploy

4. Confirm NFS server configuration.

   Confirm whether root file system directory name installed in the host system is exported or not. You can refer to the section 4.4.3 for details.
5. Booting the target system.

   The following messages will be displayed in the boot process (through monitor or serial calbel).

```
1) Using following default configuration
serverip    = <default serverip>
nfsrootdir = <default nfsrootdir>
2) Manual configuration
```

```
x) Exit to shell
>> 1
```

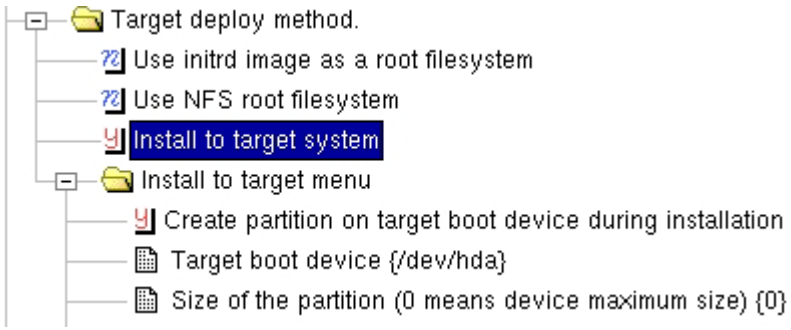
Check all values are collect, and choose ‘1’ if it does otherwise choose ‘2’.

3.7. Installing to the hard disk of target system

Using this method, you can download target root filesystem image and install it to the hard disk in the target system. After the installation, the target system can boot stand-alone.

The following steps show how to do that.

1. Select *‘Install to target system’* option



2. Execute *Build* > *Build All* in the menu.
3. Execute *Build* > *Deploy Target Image* in the menu.

   Check if the following files are created.

   /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. Check DHCPD and TFTPD are running correctly on the host system.

   Refer to section 4.4.1 and 4.4.2 for details.
5. Boot the target system.
6. The following messages will be displayed in the boot process (through monitor or serial calbel).

   Select ‘1’.

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

```
>> 1
```

7. Set the value of ‘Remote host address’ and install configuration file name (default is ‘install.conf’).

```
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
..
```

Table 1. Target Installer Ethernet install configuration

8. After proper installation, push ‘R’ key in the main display and reboot. At that time, make sure the floppy disk or CD-ROM is ejected.

3.7.1. update method after installation

Execute following command in the target system.

```
# /sbin/qp_update
```

You may find detailed information in the Manuals/remote_update.txt file of Qplus-P release.

4. Deployment to Target (Arm/Zaurus)

5. Deployment to Target (Arm/iPAQ)

6. Deployment to Target (Arm/Samsung SDMK2400)

This chapter explains how to make and deploy target image for Samsung SMDK2400X (ARM920T based), and how to use its bootloader for booting. The BSP for SMDK2400 should be installed.

6.1. Boot Loader

Original boot loader for SMKD2400 provided by Samsung has very limited usage, so Target Builder provides a more powerful and flexible one. You can both replace a original bootloader with a Target Bulider's and use the new one with the original.

Target Builder BSP for SMDK2400 comprises bootloader images in *'/opt/q+esto/bsp/arm/s3c2400/tools'* directory.

netboot-0.5-ram.bin

This loader can download a target image through serial port, but built-in boot image in 24xmon package must be used.

netboot-0.5-rom.bin

This loader can download a target image through network, but new loader must be burned on flash in place of built-in boot loader.

Target Builder's bootloader is can be use to load not only Linux image but other images also.

Notice: SMDK2400 boards have no EEPROM which is used to store MAC address, it is a critical problem for using BOOTP/DHCP protocols. For this bug, you should isolate your target/host system's network connection physically from outside, otherwise DHCPD may have a severe problem in searching the board based on the MAC address.

6.1.1. Running netboot from RAM

Using this method, you can download Target Builder's bootloader with the original one.

Do as the following steps.

- Run "dnw.exe" program which is included in SMDK2400's package you bought. (windows host is required. Sorry~~~).
- Set the COM port configuration to <COM1, 115200 baud rate>.

- Power on and boot your target board

- After “dnw.exe” program completes test of target memory, select “netboot-0.5-ram.bin” in ‘*Serial Port/Transmit’* Menu to load new boot loader image into ram.

Figure 1 shows the screen of above procedures.

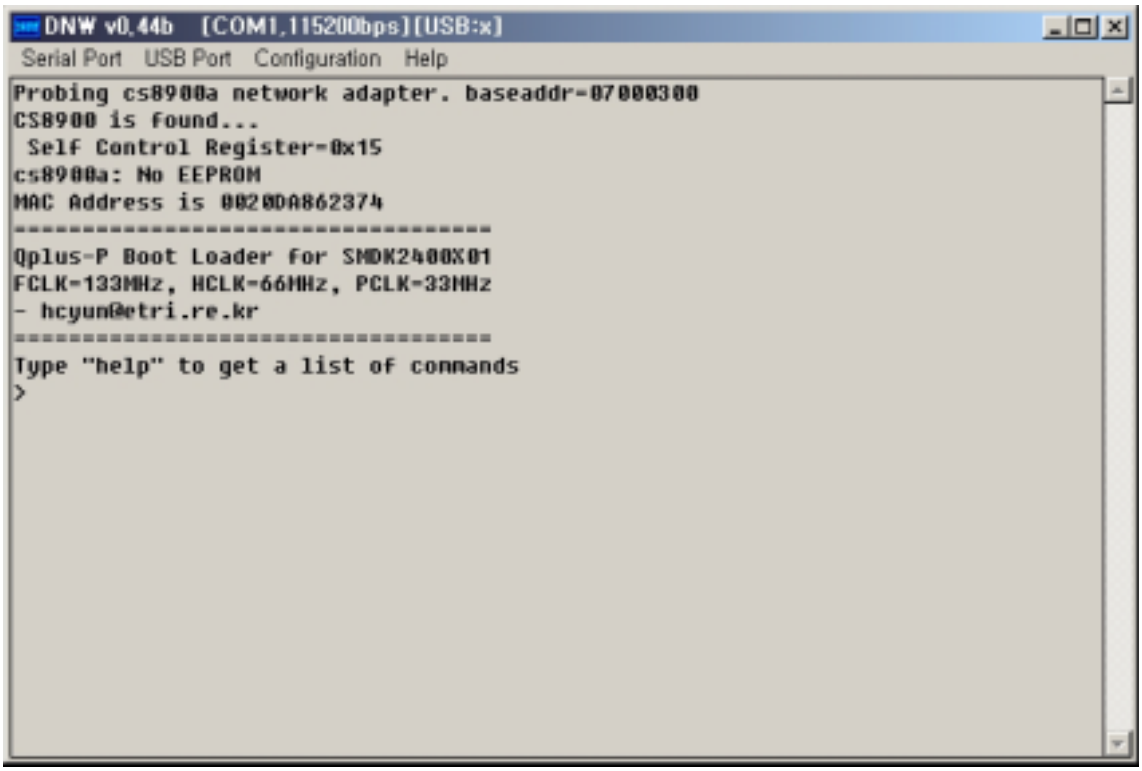


Figure 1. bootloader

Here are list of bootloader’s commands.

help

boot

Jump to and run the address which is saved in ‘kernel-addr’ environment variable.

dn <filename> <address>

Downloads host’s <filename> to target’s <address>.

tftp

Download files of name in ‘kernel’ and ‘ramdisk’ environment variables, and locate them to the address in ‘kernel-addr’ and ‘ramdisk-addr’ environment variables respectively.

It does the same as,

Dn <kernel.> <kernel-addr>

Dn <ramdisk> <ramdisk-addr>

Jump <address>

jump to <address>

printenv

Print all or part of environment. The following is default setting value for boot loader

setenv <param> <value>

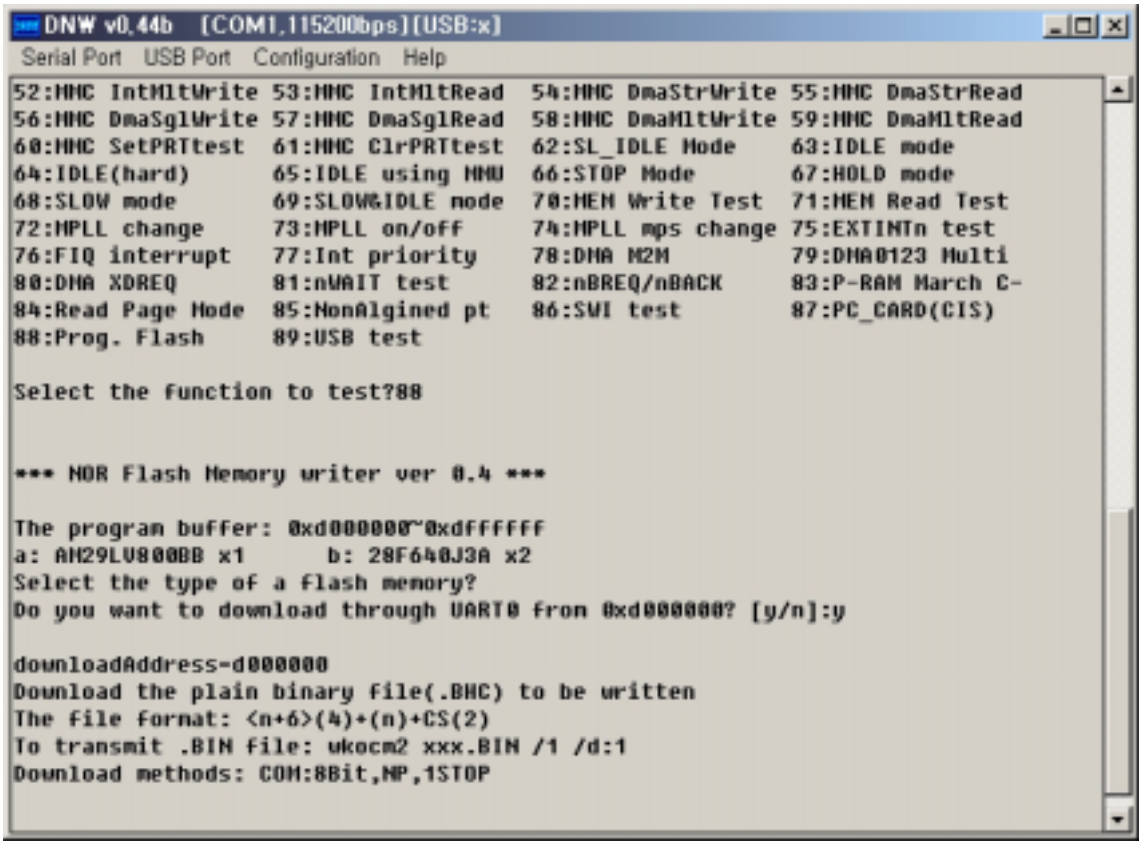
Changes or adds an environment variable.

6.1.2. Running netboot from flash (Programming flash memory for new boot loader)

If you wish to use new boot loader in stead of built-in loader, you erase old boot loader from flash memory and rewrite “netboot-0.5.rom.bin” onto flash using 24xtest’s module. We recommend that you to use a new boot loader. Because new boot loader modify and improve built-in loader in order to support networking. If you want to keep old loader, skip this chapter and refer to section 4.1.1.

The steps of burning flash is as follows

1. Power-on, boot your target board
2. Download “24xtest.bin” using DNW
3. Choose ‘Prog Flash’ .



4. Download “netboot-0.5-rom.bin”

   Write will continue to the address of 0x80000

   Notice: You must select AMD flash type. Netboot only works correctly in AMD flash type

5. Reboot target board

6.2. Deployment with Initrd root

Now, all setup required for Target Builder was finished. From this section, we will explain how to configure diverse method of deployment using Target Builder.

Initrd is a ramdisk image of target root filesystem. Because ramdisk has size-constraint of 4M~8M, deployment using initrd root filesystem is useful for small target system(in memory size).

The follow steps show how to deploy using initrd

1. Choose ‘*Use Initrd as a root filesystem’* option.

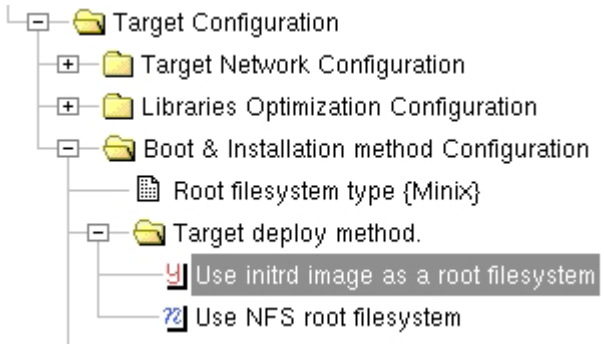


Figure 2. Inird deployment option

2. Choose *"Build > Build All"* in the menu.

   Although execute 'Build' in kernel or application configuration, execute 'Build All' in the menu again, because kernel or application program configuration can be changed by deploy option. Following figure shows you dependency rules related to *'Use Initrd as a root filesystem'* option. If you select initrd deployment method, *'RAM disk support'* and *'Initial RAM disk (initrd) support'* option in the kernel will be enabled automatically.

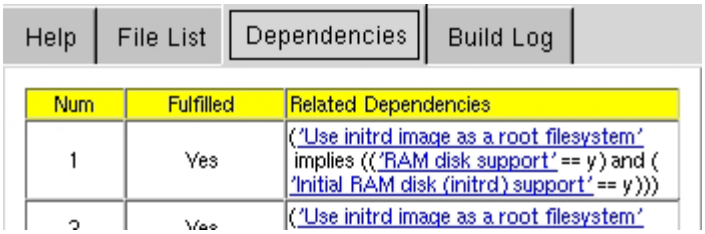


Figure 3. Automatic dependency checking related initrd

3. Choose "Build > Deploy Target Image"

   After previous step completes, this deployment method creates kernel image and ramdisk image in "/tftpbot" directory.

   /tftpboot/ss-kernel ← kernel image

   /tftpboot/ss-ramdisk ← ramdisk image

   Notice: The size of ramdisk supports upto 8M. if you want to use a larger image (more than 8M), you must use NFS deployment method,

Figure 4. Initrd deploy log

4. Check DHCPD and TFTPD are working on you host system.

   Refer to section 4.4 for the setup of the services.

5. Power on, boot your target board to starting boot loader program

   For more information of operating Netboot , refer to chapter 6.1

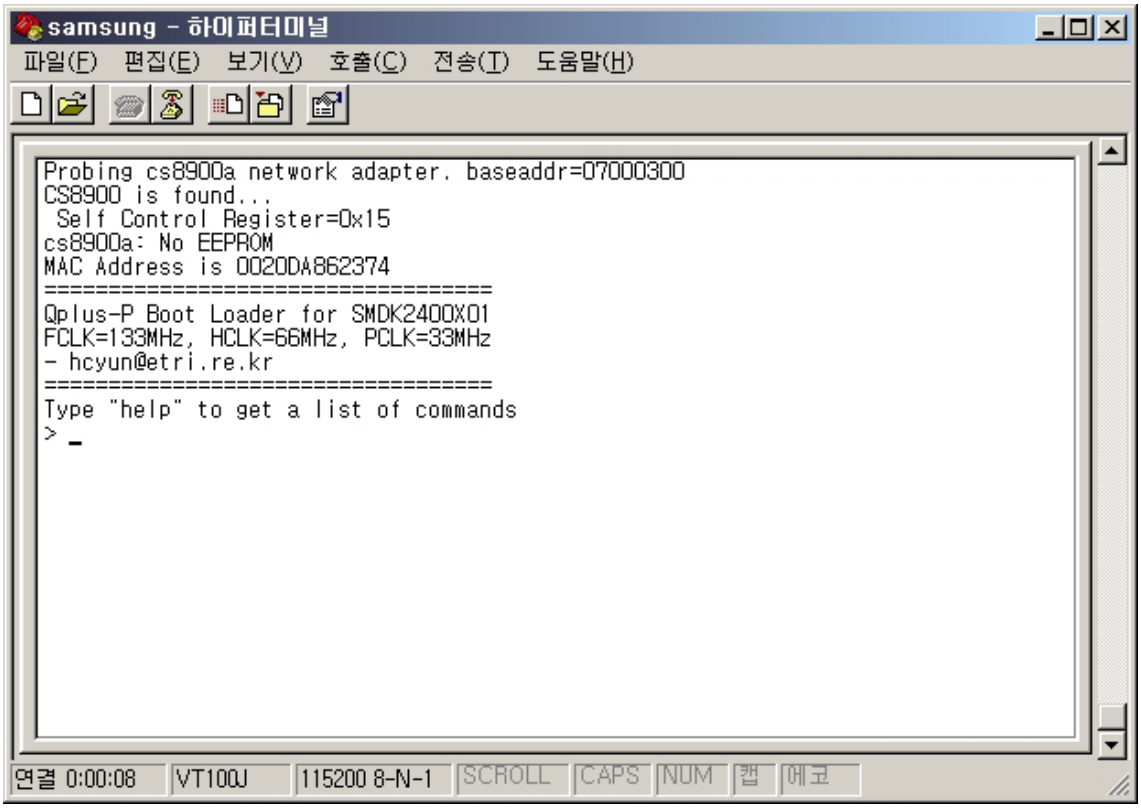


Figure5. The screen of starting boot loader

6. Check addresses and file names using ‘printenv’ command

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

> _
```

The above picture shows that ss-kernel and ss-ramdisk are going to be downloaded at kernel-addr(0x0cf00000) and ramdisk-addr(0x0c800000) respectively.

7. Type “tftp” command

```
-----------------------------------
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. Type ‘boot’ command

   After all steps are completed, you will see the following screen of ‘QPlus-P’

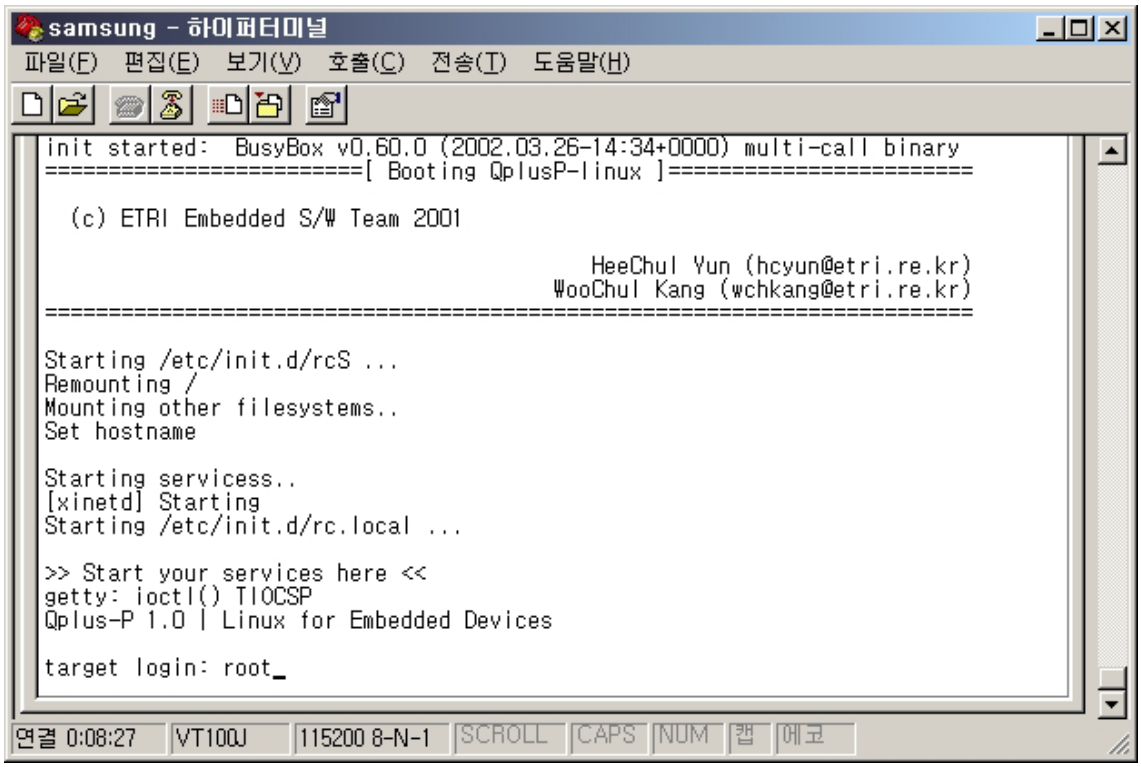


Figure 6. Initial Screen of QPlus-P after successful booting

6.3. Deploy with NFS root

This deployment method uses a particular directory of a host system as a root directory of a target system via network connection. This a very useful method in the development stage, because you can add, delete and modify files in the host system easily.

The following steps show how to deploy using NFS root filesystem.

1. Select *‘Use NFS root filesystem’* option, and then select sub-options (nfs server configuration).

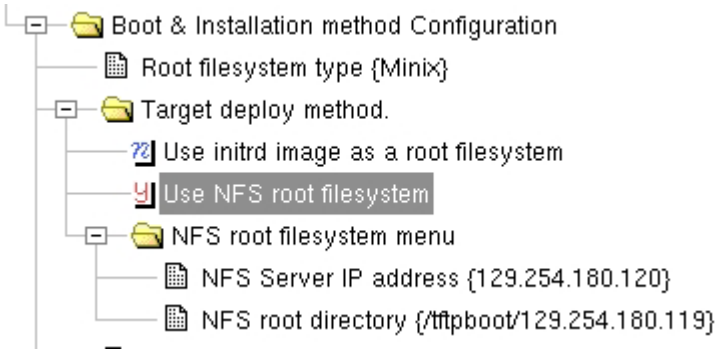


Figure7. nfs deployment option

If you select *'Use NFS root filesystem'* option, two optons, *'Syncing targetnfsroot with project'* and *'NFS root filesystem'*, will come out. *'Syncing targetnfsroot with project'* option makes Target Builder to synchronize the contents of newly generated target root filesystem(in <projdir>/target/rootfs) and exported NFS directory(including sub directories) using rsync command. If the option is not selected, Target Builder generates only a tar archive file with the generated target root filesystem.

2. Choose *Build* > *Build All* in the menu.
3. Choose *'Build* > *Deploy Target Image'*

   NFS deployment creates the following files in /tftpboot directory.

   /tftpboot/ss-kernel ← kernel image

   /tftpboot/ss-ramdisk ← ititrd image for booting via NFS

   /tftpboot/rootfs.tar.gz ← target's root file system (compressed form)

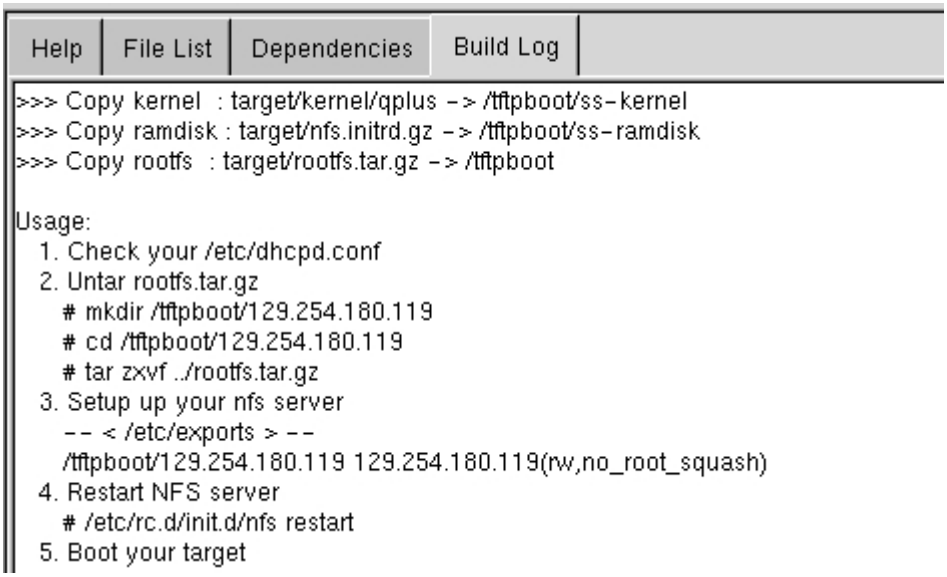


Figure 8. Build Log Screen of NFS deployment

The above figure shows log of deployment via NFS. This also indicates what to do next, just follow the steps.

4. Install rootfs.tar.gz.

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

6. Confirm NFS server configuration.

   Check whether root file system directory name installed in the host system is exported or not. You can refer to the section 4.4.3 for details.

7. Power on, boot your system to run boot loader program

   For more information of operating Netboot , refer to chapter 6.1

8. Type ‘tftp’ command

9. Type ‘boot’ command

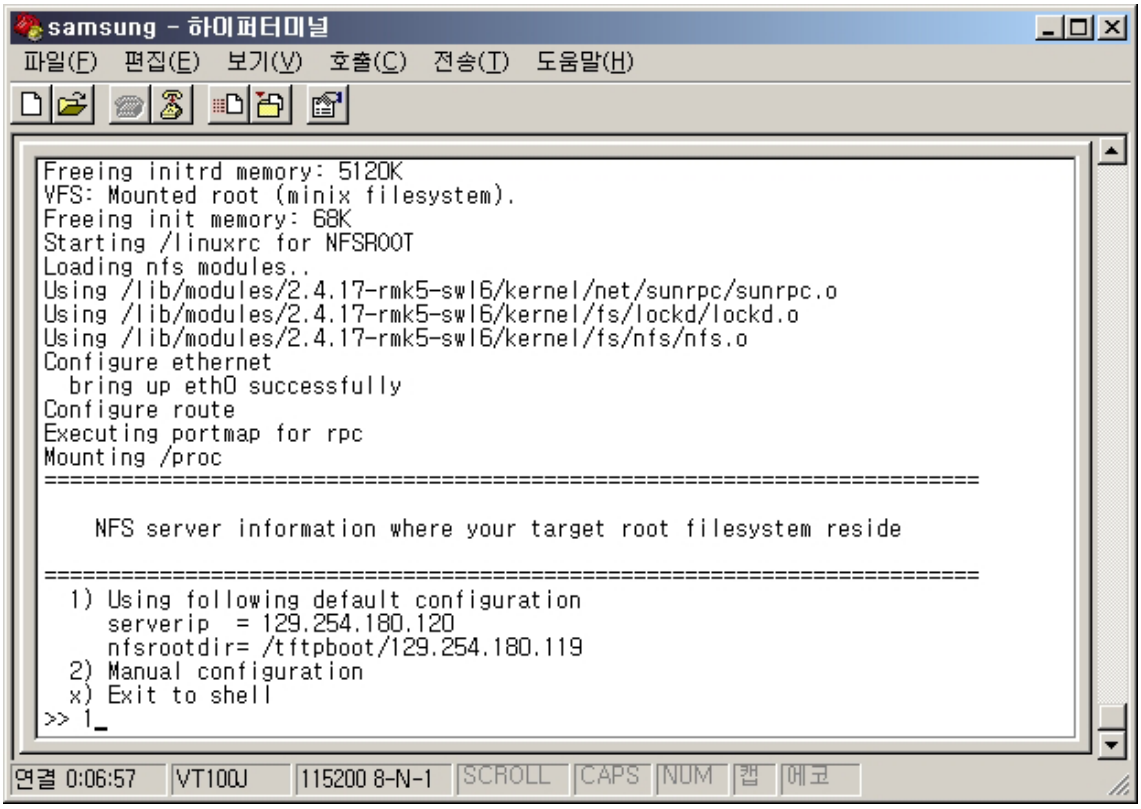


figure9. The screen of booting via NFS

If "1" is chosen, NFS services use default configuration which already specified using Target Builder. If you choose "2", you can set up manually NFS services related configuration files. If you choose other number "1" or "2", you will exit to the shell prompt.

6.4. Setting up host server services

Refer to section 4.4.1 and 4.4.2 for details.

7. Adding a custome applications to your project

Currently, our Qplus-P Target Builder supports about 60 basic application program packages. However, these packages are only basic packages required to operate linux. Actual embedded system development requires for us to configure and build more applications on Qplus-P Target Builder. Target Builder allows you to place custom applications and configuration files on your target. This chapter explains how to add a custom application to Target Builder.

There are following two method.

- The first method is a simple way of getting your custom files onto a target. In this case, the user directory contains custom files to be merged into the target image in the target directory after the Target Builder's default packages have been placed there.
- The second method is to integate your packages into Target Builder so that they can be cofigured and built alongside the Target Builder's default packages.

7.1. Merging your files to the target image

Target Builder generates temporary target root file system in the directory "*<project directory>/target/rootfs*". This temporary root file system is changed into loadable form and installed onto the target.

Target Builder supports the following process to combine user application programs with this temporary root file system.

1. **Locate your files "<project directory>/target/usersupp" directory.**

   Note that Target Builder considers the directory "<project directory>/target/usersupp" to be root directory of actual target system. For example, if you want to install the file "test.sh" in the directory "/usr/bin/" of target, you should locate it in "<project directory>/target/usersupp/usr/bin".

2. **Select *'Build > Build All'* menu**

   Target Builder merges the files in '<projdir>/target/usersupp' into the '<projdir>/target/rootfs' directory.

   Warning: Note that the when the actual files are loaded into the target root file system, ownership and permission are transmitted as they are.

7.2. Adding a package into Target Builder

Merging directories works well when you have a fixed set of files to place on your targets. However, this method doesn't provide a way for you to maintain all configuration under the Target Builder's project state.

Target Builder was designed to allow you to configure individual applications through a simple interface. If you integrate your applications into Target Builder, you gain the advantage of begin able to configure your application alongside the default packages.

7.2.1. Components of a package

Each Target Builder package consists of the following files.

file	usage
SRPM file	Contains source code and spec file ot the package
QPD file	Contains information relevant to Target Builder

SRPM file consists of compressed sources and spec file, which explain how to compile and install them, and generated by '*rpm*' command. To generate SRPM, run "rpm –ba <package spec file>" after locating spec files and compressed source files to the designated location(in case of Redhat Linux, "/usr/src/redhat").

Note: refer to http://www.rpm.org/max-rpm for detailed information on RPM.

QPD(Qplus Package Descriptor) is a file that has package information. To add packages, you should describe on it how to configurable items of each package.

7.2.2. Creating a QPD file

QPD file format is a simple extension of RPM spec file. You can create it by adding options at the end of the pakcage's RPM spec file.

QPD file = rpm spec file + additional package information

Grammatical syntax of QPD file

The following is syntax of QPD file. QPD files should be created according to the following rules.

```
QPD ::= <common spec file section>   <package information>
```

;; QPD file starts with the tag, '%package'
;; Package information is a series of options.
<package information > ::= '%package' <package name>
<option subfield >
[<option> | <group>]*

;; Each option have name and property
<option> ::= '%option' <option name>
< option subfield >

;; Group is a option that includes only simple description.
<group> ::== '%group' <group name>
<group subfield>

;; Each subfield indicates properties of the option
<option subfield> ::== <prompt>
[<export symbol>]
[<files>]
[<require>]
[<provide>]
[<build_vars>]
[<help>]

<group subfield> ::== <prompt>
[<help>]

;; Simple description about option
<prompt> ::= '%%prompt' <string>

;; Detailed description about option
<help> ::= '%%desc' <string>

<build_vars> ::= '%%build_vars' <string>

;; In case of option being chosen, macro symbols to be written on external

```
header file
<export symbol> ::= '%%export_symbol'   <external symbol> *

;; In case of option being chosen, list of files to be installed onto the target
<files> ::= '%%files'   <file name>*

;;Dependency rules
<require> ::= <logical>

;;To remove confliction between options
<provide> :: = '%%provide'   <symbol>*

<option name>, <package name > :: = [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> ')'
```

%package, %group and %option

QPD file have three types of item, %package , %group, and %option to describe property of each application package.

The item ***%package <package name>*** indicates the start of QPD's own region, which describes overall property of package and appears once for each QPD file. The item ***%option <option name>*** may have various sub-fields, which describes the option's properties. The item ***%group <group name>*** is one sort of %option, which can be used to group related options. %group can have only %%desc and %%desc properties.

Hierarchical and naming convention of options

Package, group and options have tree structure. Each package includes option and group, which can recursively include another group and option. This recursive structure is expressed in the form of tree.

Group and option is named for tree

Each item's name is used to show where it belongs to. Each item's name should include all names of its parents to the root item and they are separated using '/'. For example, in case that the package "foo"

includes the option “goo” which also includes the option “hoo”, it can be expressed as followings

```
%package foo              <= the package “foo”
…
%option foo/goo           <= the option “goo” belongs to the package “foo”
…
%option foo/goo/hoo       <= the option “hoo” belongs to the option “goo”
```

Warning: Note that only alphabets and numbers are used for the name of package, option, and group and they are case insensitive, because internal CML2 engine requires that.

Properties

Each item(package, option, and group) has its own property. Those properties are used (1)to inform the user what the item is for, (2) to make a file list to be installed to the target when the item is selected.
Each items can have the following properties.

name	Description	applicable item
%%prompt	Brief description for the item	all
%%desc	Long description for the item	all
%%files	Files to be installed to the target system if the option is selected	package, option
%%require	Dependencies of of the item	package, option
%%provide	Remove confliction between items. Items which provide the same symbol can’t chosen together.	package, option
%%export_symbol	Used only for busybox and tinylogin	package, option
%%build_vars	Patch specific strings with another string if the option is selected.	package, option

Properties can be added to each item without order. For example, in case that the option “foo/goo” has properties of “%%prompt, %%desc, %%files, %%require”, it is described as follows.

```
%option      foo/goo          <=  the option “goo” belonging to the package “foo”
%%prompt     some special option     <=  brief description
%%files      /usr/bin/good-file      <= Three files are installed on target
```

```
                    /usr/bin/bad-file
                    /etc/goo.conf
    %%require      foo/hoo ==y      <= required for the option “foo/hoo” to be chosen
    %%desc                          <= very long help
    it is a very very very long long long
    long~~~~~long help file.
```

File list

The property of file list describes list of files to be installed when each package/option item is chosen. The following example shows if package ‘foo’ is selected, ‘/usr/local/bin/prog1’ and ‘/usr’local/bin/prog2’ files will be installed to the target.

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

Dependency rule

Dependency of each package item can be described by ‘%%require’ property. Each item can have dependencies on kernel and other items. Target Builder check those dependencies every time you change the value of items and shows warning message what it is violated.

Dependencies are stated using a logical expression. The following operators can be used.

and

or

not:

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

Implies (represents inclusion relationship such as $\supset$ or $\rightarrow$)

To identify another package item, full path name (which includes its parents recursively) of the corresponding item is used.

The following example shows option ‘foo/goo/goo’ requires ‘haa/huu/hee’ option should be enable and ‘haa/huu/hii/’ option should be disabled.

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

Avoiding conflict between options

Some options or packages should not be chosen together. In this case, you can use ‘%%provide’ property to avoid it. Two options(or packages) exporting the same symbol can’t be enabled together. For example, if both ‘busybox’ package and ‘procps’ package have ‘ps’ command and you don’t want to install them together., then you can add ‘%%provide’ property to both options as the following example.

busybox.qpd

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

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

procps.qpd

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

Above two options provide the same symbol “PS”, it can’t be chosen concurrently.

Controlling build option

Target Builder compiles according to information described in spec region of qpd file, not using spec file in SRPM. And Target Builder offers the way to control compilation options in the qpd file. '%%build_vars' property replaces specific string in the spec region of the QPD file in case of items. To do this, string to be replace should be embrace with "!!" symbol, and state in '%%build_vars" property what string will replace the embraced string if the the item is chosen.

For example, suppose that the package "foo" can be compiled statically by giving "DOSTATIC=true" option to 'make' command. Then QPD file's %build section should be changed as follows

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

And '%%build_vars' property states what string will replace "!!MYOPT!! string as follows.

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

In case that the option "tinylogin/static" of the package "tinylogin" is chosen, the string !!MYOPT!! will be replaced with the string DOSATIC=true.

Writing spec region inside QPD

The spec region of the QPD file uses the same format as is ordinary RPM spec file except the following constrains.

- **QPD doesn't support sub package**

  RPM can include sub-package using %package. However, it overlaps with the tag %package which represents start of QPD region. And also, QPD don't support sub-package concept yet. Therefore, modification is required for the spec files which uses generate sub packages.

- **Restriction on using macros**

  RPM variables is not substituted in the Preamble part of QPD file. In other words, if Name is foo, the variable %{Name} is interpreted as it is, not 'foo'. Therefore, be careful.

7.2.3. Registration of SRPM and QPD file

Locate your SRPM file in <projdir>/packages/SRPM and QPD file in <projdir>/packages/QPDS respectively. Close your current project and open it again. Now your application will appear in the Target Builder.

7.2.4. Example

In this section, we illustrate how to write out and register simple QPD file of a simple package.
More example can be seen in <project directory>/packages

We will add package ‘foo’’ to the Target Biulder as the following sequence.

Making a SRPM file

- locate the file foo-1.0.tar.gz in the directory /usr/src/redhat/SOURCES.
- write out Spec file and locate it in the directory /usr/src/redhat/SPECS.
- execute the following command with root authority.
  rpm –ba foo.spec
- foo-1.0.src.rpm was generated in the directory /usr/src/redhat/SRPMS.

Writing a QPD file

- copy above spec file and then modify file extension to foo.qpd. Then, the file will be showen as follows.

```
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)
/
```

- insert the item %package at the end of the QPD file.

```
. . .
%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
. . .
```

Two files ‘/usr/local/bin/hoo’ and ‘/usr/local/bin/haa’ will be installed to the target if ‘foo’ package is selected.

In case that you want to install the file “/usr/local/hee” optionally, add options as follows.

```
. . .
%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~~~
. . .
```

The following shows final QPD file. The boldfaced region was added to the original RPM spec file to describe configuration item in QPD.

```
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~~~

Place of QPD and SRPM files

- copy foo.qpd into <project directory >/packages/QPDS
- copy foo-1.0.src.rpm into <project directory >/packages/SRPMS

Restarting Target Builder

8. Using a Targer Builer on Terminal Environment

All basic functions of the Target Builder can be available on the shell prompt but internal function for the Target Builer GUI. That is, user can execute all development process, from project evaluation to target deployment, on the shell command environment. It shows that Target Builder has very flexible design mechanism. Full description about Target Builder structure could be found on additional system design document.

If one prefers shell environment to GUI interface or has not sufficient system resource, refer to below description.

8.1. Project Creation

Figure.1. Project creation

1. Execute ‘mkprojct’ command.

```
# /opt/q+esto/tc/bin/mkproject
```

2. Enter your project name.

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

Select a CPU architecture

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

3. Select a evlaution board name

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

After above procedures, you can see the below message about the project creation messages on your screen.

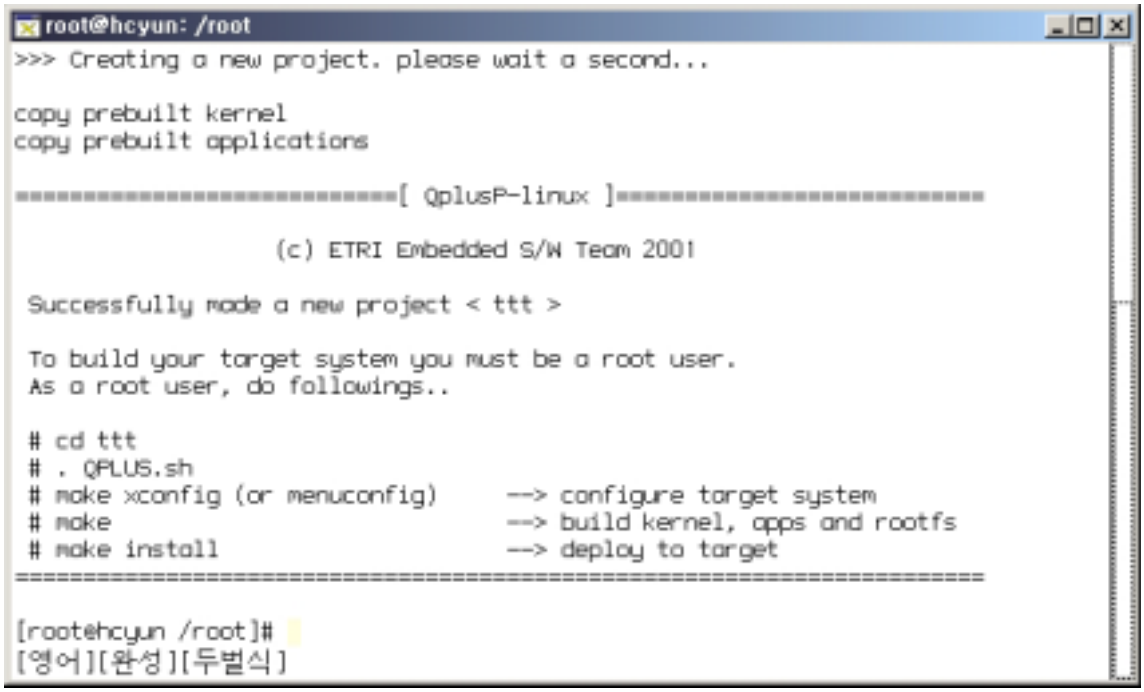


Fig. 2. The initial screen after project evaluation

4. Execute QPLUS.sh

QPULS.sh sets environment variables for Target Builder.

```
# . QPLUS.sh
or
# source QPLUS.sh
```

8.2. System Configuration

CML2 rule files can be used with any configurator which understands how CML2 rules works. Target Builder with a GUI interface is a sort of CML2 configuratior with extended functionalities. But all CML2 configurator generates result of configuration in the same file, ‘config.out’ in your project directory.

In this section we will explain other CML2 configurators, which runs in shell prompt environment.

8.2.1. menuconfig

The menuconfig configurator of CML2 was shown in Fig. 20. It has the same configuration items as the Target Builder's one. Perform the kernel, system and target configuration. Refer to Target Builder user's guide for each item.

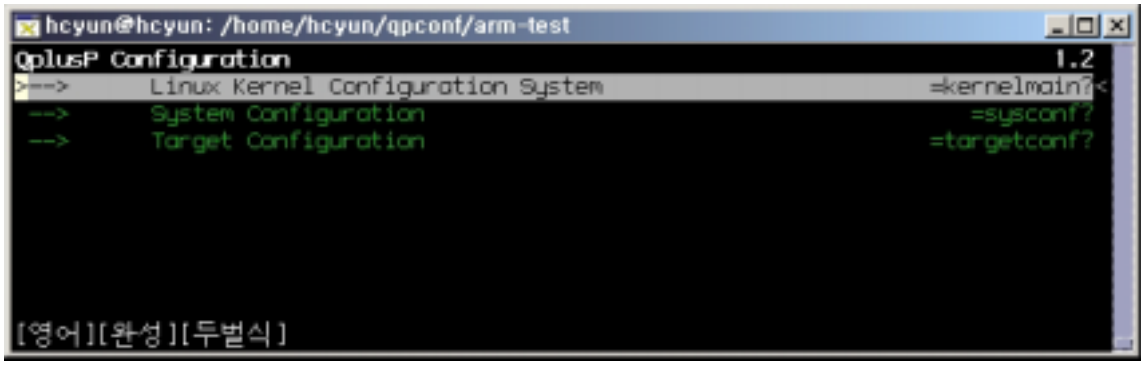


Fig.3. Menuconfig

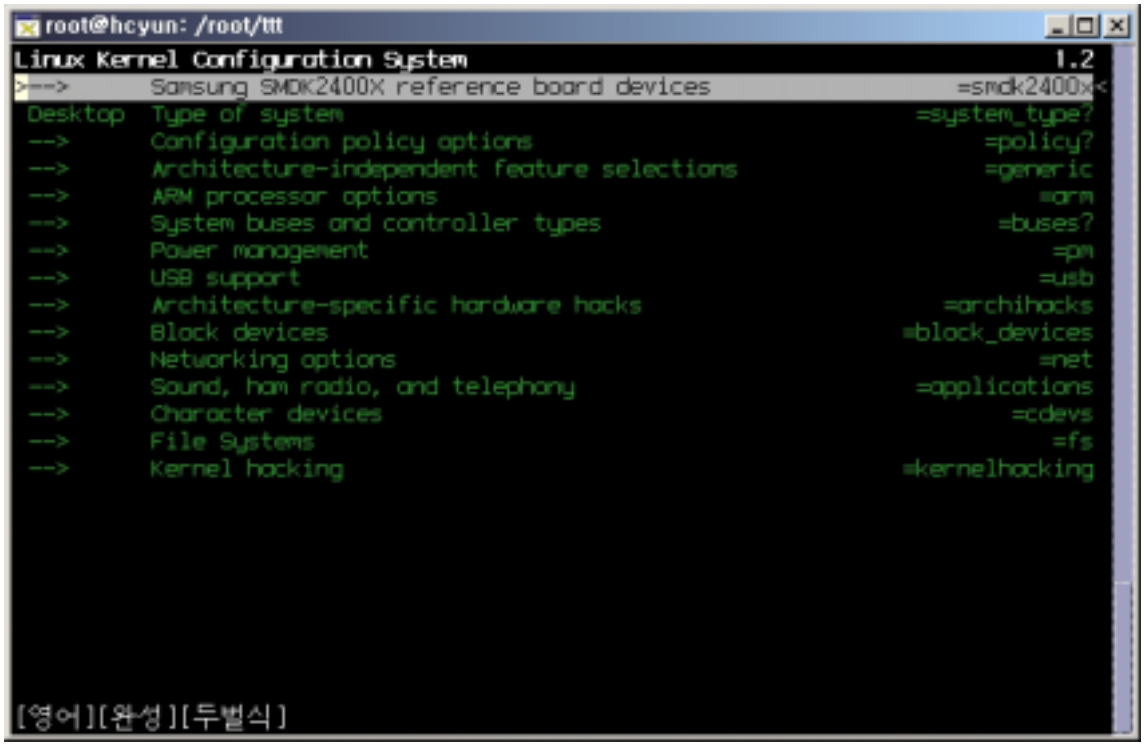


Fig.4. Kenel configuration with menuconfig

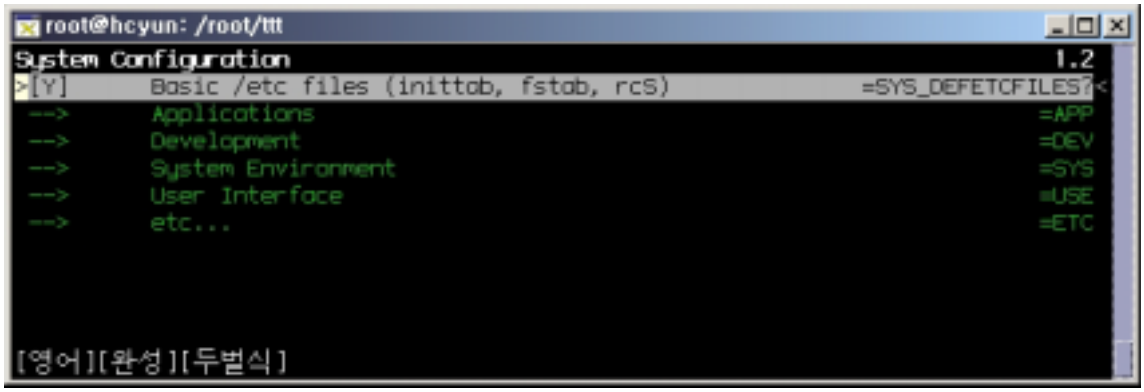


Fig.5. System Configuration with Menuconfig

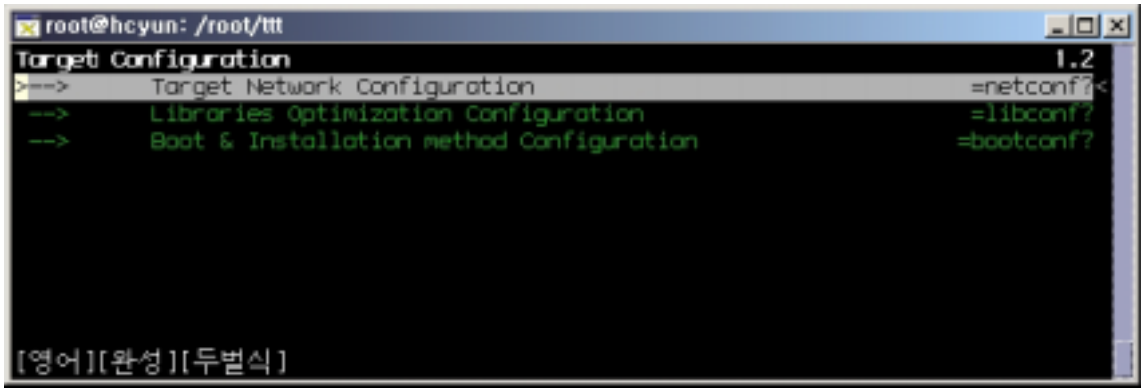


Fig.6. Target Configuration with Menuconfig

8.2.2. xconfig

Xoncifg is another TK based CML2 configuratior..All configuration items are identical.

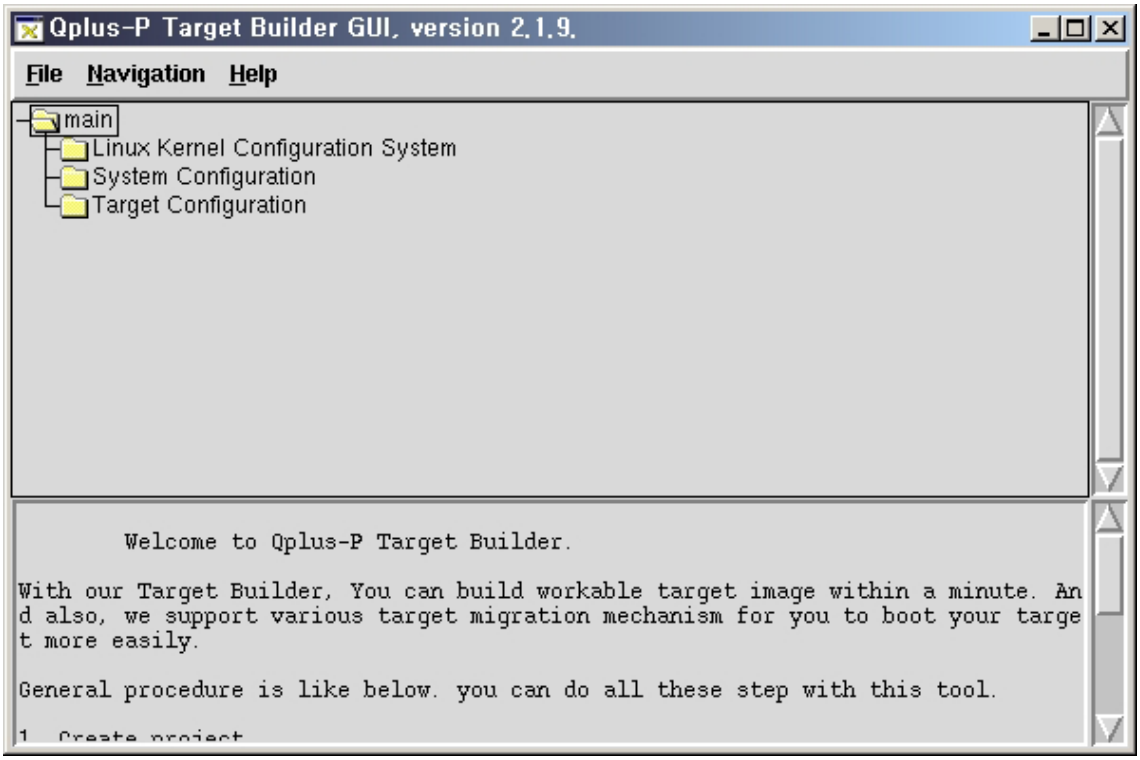


Fig.7. Xconfig

8.3. Target Image Generation

To build kernel and applications, just type as follows.

```
# make
```

This does the same job as '*Build > Build Al'l* menu in the Target Builder GUI. It compiles modified one only, so compilation does not take a long time.

Alternatively, you can build kernel and root filesystem separately as the following shows.

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

Final build result can be found in

<projdir>/target/kernel/qplus <= kernel image

<projdir>/target/rootfs.tar.gz <= target root filesystem

If you want to control the building process sophistically, you can use kernel.py, buildpkgs.py, mkrootfs.py. Usage description of these can be found by typing with '-h' option.

8.4. Target Deloyment

The method to deploy the image to the target is shown in following command.

```
# make install
```

It executes <projdir>/tools/targetinstall shell script intenally. So if you want to extend the functionality of Target Builder, you need to modify it.

If no problem, you will meet the following confirmation messages shown in Fig. 8 that describes the target booting methods. One need to read and follow the messages displayed.

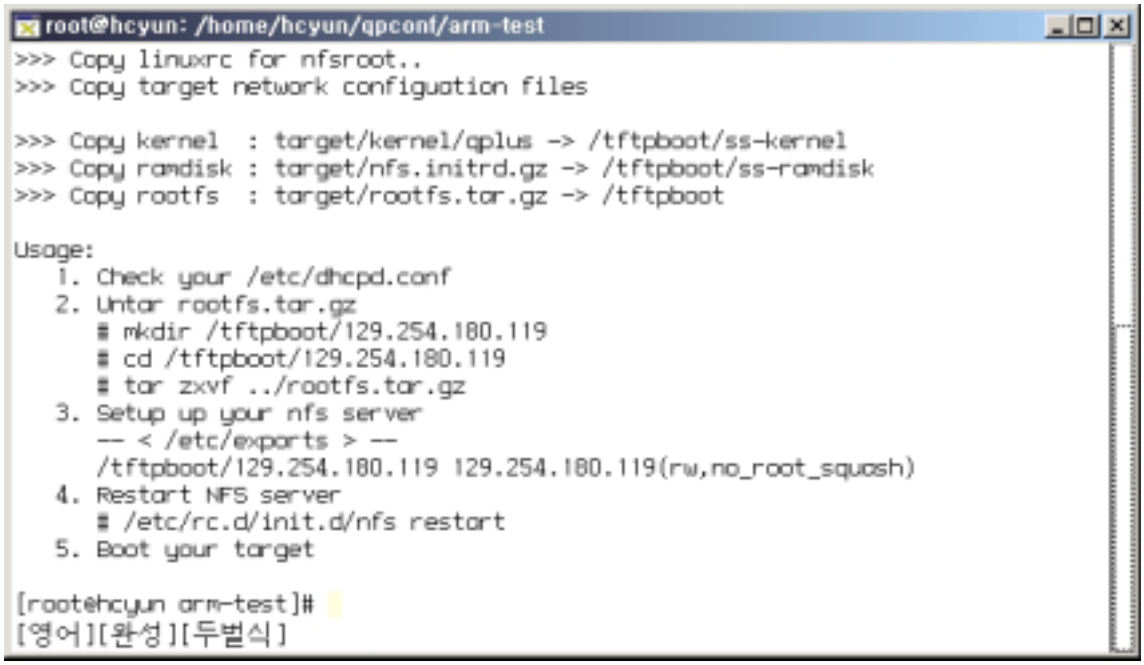


Fig.8. NFS deployment instruction