XPedite8150

XPedite8150 Linux Config. 1 BSP Manual

Revision A

X-ES
Extreme Engineering Solutions

...Always Fast

Extreme Engineering Solutions
3225 Deming Way, Suite 120 • Middleton, WI 53562
Phone: 608.833.1155 • Fax: 608.827.6171
sales@xes-inc.com • http://www.xes-inc.com

XPedite8150 Linux Config. 1 BSP Manual

Revision History

Revision	Date	Remark
A	July 12, 2016	Initial creation for 3.18.20-xes_r5 release.

Table of Contents

List of Tables

Overview

This manual describes the process of installing and configuring a Linux development environment for the XPedite8150. The software included on the Linux BSP (Board Support Package) CD-ROM provides all the tools necessary to build and boot a custom kernel, drivers, and applications. The X-ES Linux Support Package includes:

- Linux kernel source patches to support the XPedite8150 based on mainline kernel version 3.18.20
- Binary kernel images for the target platform
- Robust target file system for development

1.1 Terminology and Conventions

Below is a glossary of terminology that is used in this manual:

Table 1.1: Terminology

Term	Definition
toolchain	A collection of programming tools, such as compilers, linkers, and assemblers.
file system	In the context of embedded Linux distributions, this term is usually used to refer to a root file system that Linux uses as its boot source and userspace application environment. This can also refer to a specific kernel file system implementation, such as NFS or JFFS2.
BusyBox	Linux application that combines tiny versions of many common UNIX utilities into a single small executable. BusyBox is popular in embedded Linux systems.

Table 1.1: Terminology *(continued)*

Term	Definition
initramfs	Initial RAM file system which is embedded within X-ES Linux kernels. This is usually a minimal BusyBox-based file system that contains a few basic tools and an initialization script used to mount another root file system.
target	The single-board computer the user application is intended to run on.

Host System Setup

The host system is an x86 machine used to supplement application development for the X-ES embedded target system.

2.1 Requirements

The following items are required to create a development platform on the host system:

- An x86 machine running Linux. Fedora Core or Ubuntu is recommended for ease of use, but most any modern Linux distribution should work. Fedora Core can be downloaded free of charge from http://fedoraproject.org/en/get-fedora.
- Access to the root account for the Linux host machine
- A working Ethernet connection
- NFS server configured and enabled on host machine
- At least 3 GB of free disk space

Familiarity with basic Linux usage is required.

A TFTP server is recommended for kernel development. A TFTP server allows a Linux kernel image to be booted over a network connection via TFTP rather than re-programming it to flash each time it is changed during the development process. Most Linux distributions have TFTP server packages you can install (see ‘tftp-server’ package for Fedora/CentOS/Red Hat, or ‘tftpd-hpa‘ or ‘tftpd‘ packages on Debian/Ubuntu systems).

A DHCP server is recommended for the host system if the LAN used by the target board does not already have one. This is also available on most Linux distributions (see ‘dhcp‘ package on Fedora/CentOS/Red Hat, or ‘dhcp3-server‘ package on Debian/Ubuntu systems).

2.2 Functional Description

The host system is used for the following functions:

- NFS server for the target root file system. Application development on an NFS file system is the preferred method.
- a Linux kernel. Some applications can simply make use of the stock kernel image provided with this support package. Other applications may require recompiling the kernel to either add patches or enable additional options or drivers.
- (Optional) TFTP server for kernel images
- (Optional) DHCP server
- (Optional) Serial console access to the target. One common setup is to connect the target board to the host machine's serial port and use 'minicom' on the host machine for console access. Minicom can be used directly on the machine or connected to remotely via ssh/telnet.

2.1 Toolchain

x86_64 boards do not require a special toolchain as long as the Host System you plan to compile a kernel image with has a x86_64 core and your Host system has any Linux distribution on it.

2.3 Target File System Installation

The X-ES Linux Support Package includes a complete Gentoo-based file system built for the corei7 processing core on the X-ES target system. This file system is usually hosted via an NFS server so that the target can remotely mount it as the root file system.

1. Insert the X-ES Linux Support Package CD-ROM into the drive on the host system, and make sure it is mounted. This set of instructions will assume that it is mounted at /mnt/cdrom, but this can vary depending on your Linux distribution.

```
$ mount /dev/cdrom /mnt/cdrom
```

2. Run the root file system installation script:

```
$ bash /mnt/cdrom/linux/rootfs/install.sh
```

   Follow the instructions and be sure to read the information it displays.

The file system should now be located in `/opt/xes/devkit/corei7_fs/` on the host system.

Be sure to configure the NFS server to export this directory so that NFS clients can mount it with read-write access and without “root squash" enabled. The recommended way to do this is to add the following line to the `/etc/exports` file:

```
/opt/xes/devkit *(rw,no_root_squash,no_all_squash,sync)
```

Then, make sure the NFS server takes those changes into effect by either restarting it or running:

```
$ exportfs -ra
```

After doing this, use <NFS SERVER>:/opt/xes/devkit/corei7_fs as the NFS root path.

2.4 Kernel Source Installation (Optional)

The X-ES Linux Support Package contains the Linux kernel source code and patches supporting the X-ES target system. In many cases, recompiling a custom kernel is not required. The stock kernel image on the X-ES Linux Support Package CD contains device drivers for all on-board devices and the most commonly used features. However, to tweak the kernel configuration, use external kernel patches, or enable additional features or device drivers, install a kernel source tree for development use.

1. Insert the X-ES Linux Support Package CD-ROM into the drive on the host system, and make sure it is mounted. This set of instructions will assume that it is mounted at /mnt/cdrom, but this can vary depending on the Linux distribution.

   ```
   $ mount /dev/cdrom /mnt/cdrom
   ```

2. Extract the stock kernel source. The location of the destination directory is not important, and this can be done with any user account. These examples will assume that the kernel tree is extracted underneath the home directory.

   ```
   $ cd ~
   $ tar -Jxf /mnt/cdrom/linux/kernel/source/linux-3.18.20.tar.xz
   ```

   Note that this kernel tree is the unmodified, mainline kernel source from kernel.org.

3. Apply the X-ES kernel patch to the tree.

   ```
   $ cd ~/linux-3.18.20
   $ patch -p1 < /mnt/cdrom/linux/kernel/source/kernel-3.18.20-xes_r5.patch
   ```

4. Copy the XPedite8150 kernel configuration file to the tree.

   ```
   $ cp /mnt/cdrom/linux/kernel/config-XPedite8150-1-3.18.20-xes_r5 ~/linux-3.18.20
   ```

5. Copy the X-ES initramfs image to the kernel tree. This is a small BusyBox-based file system that gets embedded into the kernel. It provides initscripts to handle mounting various types of root file systems, and also a minimal shell to fallback on if another root file system is not available.

```
$ cp /mnt/cdrom/linux/kernel/source/initramfs.cpio.gz-corei7-1-3.18.20-xes_r5 \
~/linux-3.18.20
```

After applying the patch, a Linux kernel tree capable of building custom kernel images is present.

Alternatively, if there is an existing git tree, the X-ES patch set to it can be applied so that individual changes can be seen.

1. Update your git tree to contain the latest updates.

```
$ git pull
```

2. Fetch the ‘stable’ 3.18.20.y patches from upstream, since the X-ES patch set may assume that they are already applied:

```
$ git fetch -t git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

3. Create a new local X-ES branch based on the v3.18.20 release tag, and check it out.

```
$ git checkout -b xpedite8150 v3.18.20
```

4. Apply the X-ES patch set to the tree:

```
$ tar -jxf \
/mnt/cdrom/linux/kernel/source/kernel-3.18.20-xes_r5-patchset.tar.bz2
$ git am linux-3.18.20-xes_r5-patchset/*.patch
```

5. Copy the XPedite8150 kernel configuration file to the tree.

```
$ cp /mnt/cdrom/linux/kernel/config-XPedite8150-1-3.18.20-xes_r5 ~/linux-3.18.20
```

6. Copy the X-ES initramfs image to the kernel tree. This is a small BusyBox-based file system that gets embedded into the kernel. It provides initscripts to handle mounting various types of root file systems, and also a minimal shell to fallback on if another root file system is not available.

```
$ cp /mnt/cdrom/linux/kernel/source/initramfs.cpio.gz-corei7-1-3.18.20-xes_r5 \
~/linux-3.18.20
```

A git repository with the complete commit history should now be available.

Compiling a Custom Kernel

This section describes how to compile a custom Linux kernel from the source code installed in Chapter 2. The kernel will be compiled using the native compilers on the host system.

Note that in order to create a custom Linux kernel, it is necessary to have the target kernel source installed. If it is not installed, refer to Chapter 2 and follow the instructions to do so.

These steps will compile a kernel for the target system. It assumes that the kernel tree has been installed to ~/linux-3.18.20.

1. Change your current directory to the root of the kernel source tree.

```
$ cd ~/linux-3.18.20
```

2. Setup the native compilation environment.

```
$ export ARCH=x86_64
```

3. Configure the kernel tree for the XPedite8150.

```
$ cp config-XPedite8150-1-3.18.20-xes_r5 .config
$ make oldconfig
```

Note that the kernel configuration can also be tweaked by running **`make menuconfig`**. Also, if the board specific configuration file cannot be found, run **`make defconfig`** to create an initial configuration file.

4. Configure the kernel tree to include the X-ES initramfs image. This is a small BusyBox-based file system that gets embedded into the kernel. It provides initscripts to handle mounting various types of root file systems, and also a minimal shell to fallback on if another root file system isn't available.

```
$ scripts/config --set-str CONFIG_INITRAMFS_SOURCE \
initramfs.cpio.gz-corei7-1-3.18.20-xes_r5
$ make oldconfig
```

The above commands will modify the .config file to point to the X-ES initramfs image for inclusion into the kernel, and then configure the tree to reflect this change to .config. You may be prompted to specify a few configuration options when running "make oldconfig". You

can normally accept the defaults by pressing <enter>. If prompted for a type of compression to use for the initramfs, choose gzip compression.

5. Build the kernel.

```
$ make
```

It may take a while to build the kernel the first time.

After the build finishes, there should be a usable kernel image in `arch/x86/boot/bzImage`. Now that a kernel has been successfully compiled, you should be able to modify kernel source files and simply run **`make`** to re-build without repeating all of the above steps.

See Chapter 4 for details on booting the new kernel.

Booting Linux on the Target

This chapter describes the process of configuring and booting Linux on the target system. The following sections describe the two X-ES supported methods of booting: using GRUB (Section 4.1) and using PXE (Section 4.2).

For both booting methods, it is assumed that you are able to boot into the coreboot bootloader on the target. Consult the Quick Start Guide or XPedite8150 User's Manual for information on getting the board powered on and getting access to the bootloader.

4.1 Local Booting

Booting Linux locally requires GRUB to be installed on the on-board NAND flash or on a USB drive connected to the board. New XPedite8150 hardware from the factory will *not* have GRUB installed in the on-board NAND flash. Instead, Linux must be installed. To make this task easier, the Linux CD can be used as a bootable medium for installing the Linux root file system and GRUB to the on-board flash.

4.1.1 Installing from Bootable Media

The Linux CD has been prepared in a way that allows booting directly from the CD using a CD-ROM drive attached to one of the available interfaces or from a removable flash device that has had the CD image copied to it. This flexibility allows installation even in the case of an external CD-ROM drive being unavailable.

Making a removable flash device bootable requires access to a Linux machine with read-write permissions to the intended flash device and access to the source CD-ROM device. Note that these permissions are usually only available when logged in as root or by using the `sudo` command.

Warning

The example that follows performs write operations to a raw block device. This will result in the loss of the partition table and any file systems that exist on the device. Make note of the device file name that corresponds to the flash device and that it contains no valuable data before proceeding.

For example, suppose the CD-ROM drive is `/dev/sr0` and the destination flash device is `/dev/sdz`. Install the Linux CD into the CD-ROM drive, then issue the following command to prepare the flash device:

```
$ dd if=/dev/sr0 of=/dev/sdz
```

After the command completes the flash device can be used to boot the XPedite8150.

The BIOS setup may require modification to ensure that the CD-ROM or flash device being used has appropriate boot priority. After successfully booting from the CD or flash device, the installation utility can be found in the `/root/installer` directory. The installer guides the rest of the setup process.

4.1.2 Booting Installed Kernel

Once GRUB is installed, the BIOS for the board will automatically wrap into it. GRUB will set up a menu screen to allow users to select (with the arrow keys) a kernel to boot. One of the choices on the list will have a Linux kernel and it should have “Linux” in the title. In order to change file systems or anything else that would ordinarily be on the Linux kernel command line, see section Section 4.1.3. Press enter to start booting.

4.1.3 Using GRUB Menu

The GRUB menu system provides a simple way to edit the kernel command line and edit operations to be done while booting. Be aware that any changes to menu items in this section will only be temporary. Instructions to permanently change an item can be found in Section 4.1.4.

Pressing ’e’ at the menu selection screen (over the selected kernel) will open an editing screen. This screen allows adding lines, editing lines, and deleting lines to the script by following the instructions below the box on the screen. Note that using ’ESC’ to leave the editing screen erases progress. Typing the letter ’b’ boots the newly configured script.

4.1.4 Booting Compiled Kernel

To boot a custom compiled kernel from a local storage device requires modifying the GRUB menu configuration file to add a new entry or modify an existing entry. Editing the GRUB menu configuration file requires booting into Linux on the XPedite8150 (for GRUB installations on an embedded device) or being able to edit and load files from another machine (for installations on a removable device).

If no removable device is plugged in, the device containing the GRUB installation that was used to boot Linux is /dev/sda (a soldered NAND flash device). However, be aware that as removable devices are plugged in (via USB, SATA, eSATA, etc.) the Linux device name for this device may change. Linux names devices based on discovery order which is distinct from BIOS boot order.

Once a Linux kernel is booted up or another computer is able to access the GRUB files of a removable device, GRUB can be configured to boot the compiled kernel. First, mount the partition by using a command like **mount /dev/sda1 /mnt/usb**. Once mounted, copy the compiled kernel image into the top-level directory of this boot partition (GRUB requires that the kernel be local and not be loaded over nfs or tftp).

Next, edit the GRUB menu configuration file `/mnt/usb/grub/menu.lst`. Editing this file is the way to permanently change the GRUB menu. It is possible to add more GRUB entries, edit other entries, change the default boot entry, etc. Go to the end of the file and add an entry similar to:

```
title Custom Built Linux Kernel
    root (hd0,0)
    kernel /bzImage root=/dev/nfs console=ttyS0,115200 ip=on \
        nfsroot=192.168.1.1:/opt/xes/devkit/corei7_fs
```

The third line is the kernel command line. The kernel command line arguments that get passed to Linux can be specified after the image location is declared. See `Documentation/kernel-parameters.txt` within the Linux kernel source tree for more information on these options. The example above uses NFS for the root file system, the first serial port device at 115200 baud for the console device, and automatic network interface configuration using BOOTP/DHCP.

Upon completion, save the GRUB configuration file and reboot the board. The added kernel entry should be on the list and can be booted it by selecting it and pressing enter.

4.2 Network Booting

Pre-eXecution Environment booting (PXE) is a way for x86 boards to boot over the network. It uses a DHCP server to find a PXE file system where all the required files are kept. PXE booting is useful because it is one of the few ways to boot a x86 board over a network instead of requiring a local copy of a kernel, and it is simple to set up.

The requirements to PXE boot include:

1. A DHCP server with access to its configuration file.

2. A TFTP server that can be accessed by the DHCP server and the user.

Note that the setup procedure for these servers is beyond the scope of this document.

The steps to PXE boot are broken into two major sections: setting up the PXE file system and configuring the BIOS.

4.2.1 Setup PXE file system

The DHCP server needs to be configured in order for the BIOS to query it. The config file for the DHCP server (dhcpd in this example) should have a section similar to the following example:

```
#/etc/dhcpd.conf
# x86 Boards (PXE)
group {
    next-server 10.0.0.3; # This should be the IP to your tftp server
    filename "tftp/server/path/to/pxelinux.0";

    # The next few lines should be filled in with the appropriate
    # variables within each "${VAR}"
    host XPedite8150-${BOARD_SERIAL_NUMBER}-RP1 {
        hardware Ethernet ${MAC_ADDR}; fixed-address ${UNIQUE_IP};
    }
}
```

Also, a standard PXELinux file system will need to be put somewhere the tftp server can reach it. The file system should look like this:

```
pxe/
   |-> pxelinux.cfg/
                  |-> default
   |-> pxelinux.0 (an executable downloaded from kernel.org)
   |-> kernel_image (vmlinux from CD or a compiled kernel
                         from Chapter 3)
```

Now that the dhcp server's config file has been edited, piece together the PXE file system using the following steps.

1. The pxelinux.0 is an executable file that comes with the SYSLINUX package and is open source. It acts like a bootloader and is able to boot over the network unlike GRUB. It can be found at http://www.kernel.org/pub/linux/utils/boot/syslinux/ and be downloaded with the SYSLINUX package. After extracting the files, pxelinux.0 can be found in the syslinux/core folder. Copy the pxelinux.0 to your PXELinux file system.

2. PXELinux uses a configuration file to define the correct kernel and kernel command line. Below is an example pxelinux.cfg/default file.

```
TIMEOUT 1
ONTIMEOUT relative/path/to/kernel_image
APPEND console=ttyS0,115200 ip=on root=/dev/nfs rw nfsroot=10.0.0.3:/path/to/corei7_fs
```

Note that the “ONTIMEOUT” file path is relative to top of the PXE file system.

3. Finally, copy the kernel image into the top level directory of the pxe file system. Put the kernel in a “kernel” folder within the pxe file system to keep clutter dowm in the top level directory. Make sure other references to the file (the config file) point to the correct path.

4.2.2 Setup BIOS for PXE boot

Once the PXELinux file system and DHCP is set up, it should be possible to configure the BIOS to network boot. Please see the XPedite8150 Hardware User’s Manual for more detail about enabling BIOS network booting.

4.2.3 Advanced PXE Configuration Files

Some setups of x86 boards will require multiple kernels to boot, multiple boards booting different kernels, etc. The following section explains where to find help do such things.

Although PXELinux supports having multiple config files, the simple example uses one config file for all devices. Please see the online PXELinux documentation if a more complex setup is required.

PXELinux also supports interactive configs that allows selecting from a list of kernels and kernel command lines at boot time. This is also described at the SYSLinux wiki page.

X-ES Linux Usage

This section explains the basics of X-ES Linux usage and how to access the various devices on the target board in Linux. It is assumed that the user is reasonably experienced with the Linux operating system, but this chapter will explain knowledge required that is X-ES-specific. The X-ES Linux file system is based on the Gentoo Linux distribution . Gentoo Linux is a good choice for embedded Linux applications as it allows a high degree of customization and source-based package installations. It is recommended that the user refer to the Gentoo Linux documentation available online for general distribution issues.

5.1 Logging In

When prompted for login, enter “root” as both the username and the password. The initramfs file system should not prompt for a password. You may change this password with the following command: **passwd root**

Some applications may require that a root password exists.

5.2 Ethernet Devices

Support for on-board ethernet devices should already be compiled into the kernel. Run the following command to see a list of all available network interfaces: **ifconfig -a**

Keep in mind that some ethernet devices will be named rpX (rear port X), bpX (backplane port X), fpX (frontpanel port X), or cpX (COM Express port X) instead of ethX (where X is a port number). Off-card ethernet devices such as a PMC or XMC ethernet card will be named using the standard ethX convention.

In the initramfs file system, you can bring up an ethernet interface using DHCP with the **ifup <device>** command. For example, to configure eth1 via DHCP, run **ifup eth1**.

5.3 Remote Access

The NFS root file system includes a ssh (secure shell) server that is enabled by default. You should be able to ssh into the target for remote access. The initramfs file system includes a telnet server which also provides remote access.

5.4 I²C

The primary I²C bus can be accessed via the `/dev/i2c-0` device. This is the standard Linux I²C character device interface. Some boards have an I²C device for GPIO that can be controlled with the **xes-gpio** program.

If the XPedite8150 is configured with multiple I²C ports, they can be accessed via the appropriate `/dev/i2c-X` character device, where X represents the desired I²C bus number.

5.5 Installing Additional Applications with Portage

The included NFS file system includes a lot of Linux software commonly needed. However, it may be necessary to install other software for development. As with other Linux distributions, there are multiple options for achieving this:

1. Obtain the application source and manually install it (via the usual **./configure ; make ; make install** for a typical Linux application package).

2. Use the Gentoo Portage software distribution system to automatically download the source, compile, and install it. This is the preferred method if possible. The X-ES file system is based on Gentoo Linux, providing access to Portage, which contains a large collection of Linux applications. For more information on Gentoo/Portage, see What is Gentoo? and the Gentoo Handbook's Portage introduction.

   For example, the following command will install the 'tcpdump' package:

   ```
   $ emerge tcpdump
   ```

   This will obtain package information from the local portage tree in `/usr/portage`, download the source, configure and install the package, and add it to the local package database. Note that if the target does not have access to the internet, the installation will fail. To work around this, make a note of the URL it attempts to download, download it from another computer and place it in the `/usr/portage/distfiles` on the target file system. With the downloaded file in the local cache, the emerge should complete successfully. For a list of available packages, browse the `/usr/portage` directory. Synchronizing the local portage tree may also be done by running **emerge --sync**.

This will ensure that the local portage tree in /usr/portage matches the latest Gentoo tree allowing installation of newer versions of software. Gentoo's **emerge** program has many options available which are not discussed in this document. Please refer to the Gentoo documentation for more information.

5.6 X-ES Linux Utilities

X-ES maintains a package of Linux scripts and utilities specifically designed for X-ES embedded systems, known as the 'xes-utils' package. This package should already be installed on the file systems included with the board support package.

Each utility is prefixed with "xes-", so entering **xes-** into the shell, and then pressing [TAB] for the shell auto-completion will show the available commands:

```
xes-getty  xes-gpio  xes-led  xes-kernel-make  xes-showecc  xes-showtemp  xes-sysinfo
```

Most of these are self-explanatory or have help available by running the command with the "-h" flag. xes-getty is a getty wrapper to automatically detect baud rates and should not normally be run.

Frequently Asked Questions

A.1 General X-ES Linux

Q: When I try to boot my custom-compiled kernel, it fails to boot with the errors “VFS: Cannot open root device ‘nfs’” and “VFS: Unable to mount root fs on unknown-block(0,255)”.

A: This is most likely caused by the initramfs image not being included in your kernel. The newer kernels use an initramfs with a small busybox-based file system that has some basic functionality and an init script that mounts the root file system. In some kernels, the kernel would do the mounting directly.

The advantage of the initramfs approach is that you have more flexibility in userspace in booting your root file system. It allows you to do things like specify “root=ramfs” to boot to a small memory file system without using NFS. You can still specify “root=/dev/nfs”, but in this case, the initramfs image did not get included with your kernel, so the root file system cannot be mounted.

See Chapter 3 for instructions on including the initramfs image in your kernel.

Q: How do I compile an out-of-tree kernel module?

A: If you need to compile a driver or kernel module that is not present in the X-ES kernel tree, you can do so by compiling your driver with the build system pointed at the X-ES kernel tree. You should be using the same compiler to build your module that you use to build the kernel. Follow these general steps as a guide:

1. Set up a working kernel as instructed in Chapter 3. You can skip the final **`make`** step if you want — that step will actually build the kernel. The important thing here is that you get the kernel tree configured properly so that the driver has an appropriate kernel tree to reference.

2. Point your driver’s build system to the kernel directory to use as the source of the kernel headers. These instructions assume that your driver makes use of the Kbuild system. The details of this will depend on how the driver build system works, but if it’s a typical Linux driver you’ll usually see something like KERN_DIR or KDIR in a top-level Makefile.

   For example, a driver Makefile may have the following by default:

```
KDIR    := /lib/modules/$(shell uname -r)/build
## ---[snip]---
all:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

In this case, you could just override KDIR by setting it to ~/linux-3.18.20

3. If necessary, set the appropriate architecture and cross-compiler settings in your driver Makefile. If the driver build system uses Kbuild properly, things should “just work” without any further modification required (that is, it just looks at ARCH and CROSS_COMPILE variables from the environment).

Then you should be able to proceed to compile your kernel module (.ko file), which can be loaded by copying it to your file system and running **`insmod <file.ko>`** on the target.

Q: How should I profile my application to see how the CPU is utilized?

A: For profiling an individual application, gprof is a good tool to use under Linux. For profiling the kernel (what you’re probably interested in if you’re doing driver development), you would normally use oprofile. This may not be enabled in your kernel configuration already. Enable oprofile support by re-compiling your kernel with CONFIG_PROFILING and CONFIG_OPROFILE enabled:

From **`make menuconfig`**:

1. Enable “Profiling support” from the top-level
2. Enable “OProfile system profiling” which will appear below “Profiling support” once enabled

Then re-compile.

SourceForge should have sufficient information explaining how to use oprofile.

Q: When emerging a Gentoo package, the system seems to run out of memory and fails to compile the package properly.

A: This can happen with larger applications, and is caused by the fact that `/tmp` is used as a tmpfs file system by default (i.e., a RAM disk). Since the emerge process uses the `/tmp` directory while building, it can run out of space in some cases.

Try running **`umount /tmp`** before doing the emerge command. This will just make the `/tmp` file system reside on the NFS root file system rather than being a RAM disk, which is necessary when emerging larger packages.

Q: My application requires a feature or configuration option not present in the provided kernel image.

A: In general, X-ES stock kernels are configured with a minimal set of options enabled by default. All of the available drivers for the appropriate on-board peripherals should be enabled to make use of the hardware, but we do not enable support for other devices to keep the kernel size down.

However, X-ES Linux board support packages do include the full source tree and default kernel configuration files so that customization is possible. Many customers have successfully applied outside kernel patches or customized the kernel configuration for their application.

Q: What software is the X-ES Linux board support package derived from?

A: The Linux kernel is based on the mainline kernel from kernel.org with some customizations and patches applied by X-ES.

The userspace distribution is a custom file system derived from Gentoo Linux. Gentoo is essentially a “versionless” distribution, in that you don’t typically have a set distribution version with a fixed set of package versions. Software package versions are tied to the Gentoo portage tree.

The development tools consist of a standard GNU toolchain (gcc, gdb, as, etc).

Q: How is the X-ES Linux board support package licensed?

A: Extreme Engineering Solutions holds a copyright on the Linux Support Package CD to the extent of the original work present on the CD (similar to any other Linux distribution’s copyright status). Original work on the CD includes user manuals, portions of the boot loader firmware, example applications, X-ES kernel modifications, and packaging of the userspace file systems.

Of course, the Linux kernel and file systems are derivative works and are built on top of the work of hundreds of copyright holders. Their copyrights still apply and all X-ES kernel code is licensed under version 2 of the GNU GPL.

The user manuals on the CD are X-ES Competition Sensitive - Do Not Share.

Q: How can I directly access registers or PCI memory from userspace?

A: On Linux, this is normally done by opening up `/dev/mem` and using the mmap() call to map the physical memory region you want. The following example application demonstrates this method in a generic way, and makes a useful command line tool for reading and writing physical addresses.

From Free Electrons:

```
/*
 * devmem2.c: Simple program to read/write from/to any location in memory.
 *
 *  Copyright (C) 2000, Jan-Derk Bakker (J.D.Bakker@its.tudelft.nl)
 *
 *
 * This software has been developed for the LART computing board
 * (http://www.lart.tudelft.nl/). The development has been sponsored by
 * the Mobile MultiMedia Communications (http://www.mmc.tudelft.nl/)
 * and Ubiquitous Communications (http://www.ubicom.tudelft.nl/)
 * projects.
 *
 * The author can be reached at:
 *
 *  Jan-Derk Bakker
 *  Information and Communication Theory Group
 *  Faculty of Information Technology and Systems
```

```
 *  Delft University of Technology
 *  P.O. Box 5031
 *  2600 GA Delft
 *  The Netherlands
 *
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <signal.h>
#include <fcntl.h>
#include <ctype.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/mman.h>

#define FATAL do { fprintf(stderr, "Error at line %d, file %s (%d) [%s]\n", \
  __LINE__, __FILE__, errno, strerror(errno)); exit(1); } while(0)

#define MAP_SIZE 4096UL
#define MAP_MASK (MAP_SIZE - 1)

int main(int argc, char **argv) {
    int fd;
    void *map_base, *virt_addr;
 unsigned long read_result, writeval;
 off_t target;
 int access_type = 'w';

 if(argc < 2) {
  fprintf(stderr, "\nUsage:\t%s { address } [ type [ data ] ]\n"
   "\taddress : memory address to act upon\n"
   "\ttype    : access operation type : [b]yte, [h]alfword, [w]ord\n"
   "\tdata    : data to be written\n\n",
   argv[0]);
  exit(1);
 }
 target = strtoul(argv[1], 0, 0);

 if(argc > 2)
  access_type = tolower(argv[2][0]);


    if((fd = open("/dev/mem", O_RDWR | O_SYNC)) == -1) FATAL;
    printf("/dev/mem opened.\n");
    fflush(stdout);

    /* Map one page */
```

```
   map_base = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, target & ~MAP_MASK);
   if(map_base == (void *) -1) FATAL;
   printf("Memory mapped at address %p.\n", map_base);
   fflush(stdout);

   virt_addr = map_base + (target & MAP_MASK);
   switch(access_type) {
 case 'b':
  read_result = *((unsigned char *) virt_addr);
  break;
 case 'h':
  read_result = *((unsigned short *) virt_addr);
  break;
 case 'w':
  read_result = *((unsigned long *) virt_addr);
  break;
 default:
  fprintf(stderr, "Illegal data type '%c'.\n", access_type);
  exit(2);
}
   printf("Value at address 0x%X (%p): 0x%X\n", target, virt_addr, read_result);
   fflush(stdout);

if(argc > 3) {
 writeval = strtoul(argv[3], 0, 0);
 switch(access_type) {
  case 'b':
   *((unsigned char *) virt_addr) = writeval;
   read_result = *((unsigned char *) virt_addr);
   break;
  case 'h':
   *((unsigned short *) virt_addr) = writeval;
   read_result = *((unsigned short *) virt_addr);
   break;
  case 'w':
   *((unsigned long *) virt_addr) = writeval;
   read_result = *((unsigned long *) virt_addr);
   break;
 }
 printf("Written 0x%X; readback 0x%X\n", writeval, read_result);
 fflush(stdout);
}

if(munmap(map_base, MAP_SIZE) == -1) FATAL;
   close(fd);
   return 0;
}
```

Q: How can I install a telnet server?

A: Note that the SSH server should be working already on the full Gentoo file system. The smaller flash file systems and/or initramfs enable telnet by default. If you do want telnet capability in addition to ssh on the Gentoo file system, you can do so with the following steps:

1. Install the xinetd and telnet server packages. Run the following on the target (this requires Internet access). This may take a few minutes to download and compile the packages necessary:

```
$ emerge xinetd netkit-telnetd
```

2. Enable the telnet server through xinetd (this is the standard method in Linux):
   a. Edit `/etc/xinetd.d/telnetd` on the target file system
   b. Change “disable = yes” to “disable = no”
   c. Save your changes
3. Start the xinetd server:

```
$ /etc/init.d/xinetd start
```

4. Configure the xinetd service to automatically start on bootup by adding it to the “default” runlevel. This is the standard Gentoo method for enabling a boot service.

```
$ rc-update add xinetd default
```

5. Edit `/etc/xinetd.conf` and change “only-from = localhost” to “only-from = 0.0.0.0” to allow outside hosts to access it.
6. If you want to be able to login as root via telnet, add the following to the `/etc/securetty` file to allow up to 10 root sessions:

```
pts/0
pts/1
pts/2
pts/3
pts/4
pts/5
pts/6
pts/7
pts/8
pts/9
```

You should now be able to telnet into the target board.

Q: We are doing kernel development and frequently get kernel panics. Is there a way to reduce the reboot time after a panic?

A: Simply add the “panic=X” option to the kernel command line arguments. This is a kernel feature, and the default is usually set to 180 seconds.

Q: The log files in `/var/log/` are growing to very large sizes. How can we reduce the space consumed by these log files?

A: Other than disabling syslog or re-directing the log files to `/dev/null`, this is typically handled with logrotate under Linux. Gentoo supports this via the “logrotate” package which you should just be able to emerge. Instructions are at the HOWTO page. You may want to use the “compress” option so that old logs are compressed using gzip.

Q: How can I run commands automatically on bootup?

A: There are many ways to accomplish this, but the preferred method in Gentoo Linux is to place your commands in the `/etc/conf.d/local.start` file. These will get executed by the Gentoo “local” service during bootup, and will usually happen at the very end, right before login.

Q: How do I set the timezone in my newly extracted filesystem?

A: It is important to set the timezone in your Gentoo filesystem to a meaningful value. By default, the timezone is set to a placeholder “Factory” timezone:

```
# date
Thu Jul 17 21:32:22 Local time zone must be set--see zic manual page 2014
```

1. Find an appropriate timezone in the output of the following command:

```
# find /usr/share/zoneinfo/ | sed s+/usr/share/zoneinfo/++
```

2. Suppose that we want to use the “US/Central” timezone. Write the timezone to `/etc/timezone`:

```
# echo US/Central > /etc/timezone
# emerge --config timezone-data
```

3. Verify the timezone is set:

```
# date
Thu Jul 17 16:51:32 CDT 2014
```

A.1 X-ES Linux for Core i7-based Products

Q: How do I perform a cable diagnostic test on an Intel I210 copper ethernet port?

A: Assuming your board has an I210 placed, the cable diagnostics test can be run by issuing the following command:

```
# ethtool -t <device>
```

Where <device> is the name of the target I210 copper Ethernet port.

XPedite8150 Linux Config. 1 BSP Manual

Rev. A

X-ES Competition Sensitive - Do Not Share

Extreme Engineering Solutions, Inc.
3225 Deming Way, Suite 120 • Middleton, WI 53562
Phone: 608.833.1155 • Fax: 608.827.6171
sales@xes-inc.com • http://www.xes-inc.com