Freescale Semiconductor
Application Note

Document Number: AN2579
Rev. 1, 11/2006

Porting Linux® to the MPC8260ADS

by Michael Johnston and Nigel Dick
NCSD Applications
Freescale Semiconductor, Inc.
East Kilbride, Scotland

This application note examines the historical development of the Linux® operating system and the features that make it suitable for porting onto a Freescale PowerQUICC™ II processor core. This document also describes how to set up a Linux environment on the Freescale MPC8260ADS development board and gives an example that shows the Linux kernel running.

Linux is taking up a major amount of column space in many computing journals and periodicals. In the embedded arena, Linux and the GPL applications used with it are growing at an equally rapid rate. Earlier operating systems often caused developers great frustration because they were forced to tolerate system bugs. Giving developers the power to identify and solve bugs, Linux users can write customized software patches that are incorporated into the software library for the benefit of all Linux users. The growth of Linux in the embedded marketplace is undoubtedly due to this open nature. Developers can quickly modify the Linux kernel code to suit their own particular applications.

Contents

1 Linux Basics

The Linux operating system is named for its inventor, Linus Torvalds. Dissatisfied with the operating system for his new IBM 386 PC hardware, he wanted to run a UNIX operating system instead. Because UNIX was expensive, he set out to develop a new version. From this work, a new core operating system or kernel named Linux was created. In early spring of 1994, three years after development first started, the first publicly-released version of Linux (version 1.0) became available.

A decision at the outset of the Linux project made the code distribution freely available. Linux had a number of features that were normally seen only on operating systems that cost hundreds of dollars per user. Features such as a demand-paged virtual memory system and the ability to multitask quickly generated interest from prospective developers.

According to International Data Corporation (IDC), in 1999 Linux had a comparatively large market share of 25 percent in the server market and was growing at tremendous speed. IDC figures further indicate that Linux growth outperforms that of Windows NT[1 ()]. The total number of Linux users is difficult to determine because licenses are not issued for individual users.

1.1 Linux, A Versatile Operating System

The varied features of Linux allow its use in a variety of applications. Linux is ideally suited to networking environments. Because it is a multi-tasking operating system, it can handle the needs of more than one user at any one time and run many different programs simultaneously. Such features make Linux perfectly adapted to the requirements of personal or networked workstations.

Memory can be made temporarily available using *memory swapping*, which means that all or part of a running application is temporarily saved to a mass storage device such as a hard disk. The virtual memory properties of Linux coupled with its multi-tasking capability and powerful file management system make it particularly suitable for file and print server applications. Linux also supports IP-masquerading, which allows a single Linux machine, or router, to control the routing of all network traffic from an entire network onto the Internet. Using this technique, a firewall can be set up and an entire LAN can be securely hidden behind a single IP address, which is the configuration that most businesses use today.

Because it is flexible and the source code for the entire operating system is freely available, Linux is a popular choice with many Internet service providers, who can quickly solve problems themselves rather than waiting for a commercial vendor to correct the problem. The result is less network down time. Indeed, Linux can be considered the backbone of the Internet because a high percentage of web servers in the world use it for its stability and networking capabilities.

The kernel structure of Linux includes an integrated packet filter that allows restriction of incoming and outgoing IP traffic, giving the kernel a high degree of security and immunity from viruses. Thus, the kernel structure of Linux lends itself well to Internet firewall security applications.

The Linux operating system can coexist with other operating systems on the same machine without causing problems. It can be installed on a hard drive partition (or floppy drive, DVD, and so on). With a boot loader (for example, LILO, GRUB, or U-boot), the user can select the appropriate operating system when booting up the system.

As embedded systems grow in size and complexity, an operating system becomes essential to simplify the system software. Linux is a general-purpose operating system that provides a stable platform that is portable to many processors. Because the kernel is highly configurable, or scalable, Linux can handle a wide range of applications.

1.2 Linux, the Embedded System

Linux is a reliable operating system that is gaining considerable ground in the embedded field. A key point is that the Linux operating system is just a core kernel, nothing more. To do something useful with the kernel and perform basic functions, additional elements or applications must be added to the embedded Linux system. These applications can range from simple command line utilities to commercial applications.

Because memory is usually limited in an embedded system, the size of the developer software should be as small as possible. Compressing the file system and customizing the kernel are two main ways to achieve this goal. An example of compressing the file system is cramFS, which allows a large file system to be fitted onto a small ROM. File sizes are limited to less than 16 Mbytes, and the maximum file system size is just over 256 Mbytes. Files are compressed one page at a time, and the utility allows random page accesses so that individual pages can be retrieved on demand from the storage ROM. Customizing the kernel allows configuration of only the modules and device drivers that are needed for a particular application.

A basic embedded Linux system requires the following elements:

- Boot loader that performs hardware initialization and software startup
- Linux kernel composed of timing services, process, and memory management

As a number of applications are added, the following features may be required:

- TCP/IP network stack to handle network accesses
- A file system that is RAM- or ROM(flash)-based
- Storage disk for storing semi-transient data and use during data swap conditions

There are usually limits on the available memory in the target device in which the kernel and applications can be stored. Linux has a number of features that an embedded system can exploit to save RAM. Consider a typical embedded Linux application that consists of the kernel and an application program. In this simple example, both the kernel and application tasks can be in memory when the system first boots up. Both tasks and kernel can be compiled and then downloaded as one image to the target at boot time. This image can be stored as a file system image in ROM or RAM. Note that the kernel and applications tasks can also be downloaded separately.

Because it can move programs to and from memory, Linux can save memory. When initialization code is downloaded to the target, this code is usually executed once to set up the target and never runs again until the target is reset. After the system successfully boots, the initialization code can be discarded or overwritten. Similarly, utility programs that run outside the kernel can be loaded when needed and discarded (or unloaded) to save memory so that the same area of memory loads in utility programs whenever the kernel requires them.

The choice of storage medium limits what code should be located and where it should be located. Using flash memory (for writeable file systems), which has a limited number of write cycles (for example, 100000), should be avoided because it can quickly cause the flash memory to wear out or fail. Similarly, if power interrupts are configured into the code, memory can easily be lost if an interrupt occurs during a write cycle. Flash memory is nearly always used as a file system to store the kernel programs as files that are loaded into RAM as required. The RAM itself is usually a medium for holding transient files.

Another important consideration for embedded systems is the need for a true real-time operating system. Because the phrase *real-time* is open to interpretation, there are hard and soft real-time Linux systems. Hard real-time systems require short deterministic response latencies to events, unlike soft real-time systems that are based on average response times. Linux-based RTOS packages are available to meet the requirements for both classes of real time. Linux can provide a few milliseconds average response time to simple problems or it can meet strict response times (using the real-time application interface RTAI) such as 15uS guaranteed response with only 5uS of jitter. [2] For more information, visit http://www.aero.polimi.it/~rtai/documentation/articles/guide.html.

Current Linux users may be familiar with the LILO boot manager that configures the kernel to take into account the available external hardware and perform the necessary initialization sequence. When the Linux kernel runs in an embedded PowerQUICC II microprocessor application, the initialization routines must be generated specifically for the target environment. This operation is performed using the GPL software U-Boot (universal bootloader).

2 Linux Kernel Download onto MPC8260ADS

This section describes how to set up, run, and debug a Linux kernel on the MPC8260ADS board. An environment is set up to allow a Linux kernel to be downloaded onto an embedded Freescale PowerQUICC II microprocessor. This example uses the MPC8260ADS evaluation board and a Linux kernel OS downloaded using U-Boot. A number of Linux ports for other Freescale microprocessors are either in development or were completed by individuals within the general Linux community. For more information, visit one of the many Linux discussion lists that are available on the web (for example, http://lists.linuxppc.org/).

2.1 Set Up the System

The system described in this document is based on a point-to-point system (that is, no separate machine for NFS) with the following setup:

- PC IP address — 192.168.1.1
- Target board IP address — 192.168.1.52
- Subnet — 192.168.1.0
- Netmask — 255.255.255.0

Take care when changing these values to ensure that the system works correctly.

A /tftpboot directory was created on the host to hold the kernel images downloaded to the board. To download the kernel image using tftp, ensure that the `/etc/inetd.conf` file has an entry for tftp (for example, `tftp dgram udp wait nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /tftpboot`).

This information applies when the Debian version of Linux is installed on the host machine. Certain steps can be omitted or modified slightly to work successfully on Redhat Linux (for example).

2.2 Install Linux Development Tools

Installing development tools to cross compile the kernel for the MPC8260ADS may be necessary. *Cross compiling* is the act of building source code on one system (the build host) into executables or libraries to run on a different host (the native host). The build host and native host may differ in operating system and/or processor type.[3] The following two examples of development tools incorporate cross compilers for PowerQUICC:

- *Metrowerks Platform Creation Suite for Linux OS*. A complete development environment including target wizard tools, CodeWarrior IDE, and board support packages.
- *MontaVista Linux Professional Edition 3.0.* A complete embedded operating system and cross development environment that incorporates a large array of cross-development tools for system and application development, processor and board support, and hundreds of deployable utilities, libraries, drivers, and other run-time components. For details, visit http://www.mvista.com/.

The example configuration in this document assumes that the development tools are installed in the `/opt` directory of the host machine. Furthermore, it is assumed that the host environment variables are modified to include an entry for the cross compiler. The following example shows how to do the modification:

1. Type the following:

   ```
   CROSS_COMPILE="/opt/CrossCompilerDirectory/bin/powerpc-linux-"
   ```

   Ensure that no spaces appear after the path definition, which can cause an error during the build.
2. Then type the following:

   ```
   export CROSS_COMPILE
   ```
3. Type `printenv` to display the environment variables.

   CROSS_COMPILE should be listed.

Completing these steps ensures that the link to the cross compiler does not have to be entered into every kernel/application Makefile that is compiled for the target. When the tools are installed, U-Boot and the Linux kernel can be compiled.

2.3 Communicate with the MPC8260ADS Board

To download the Linux kernel to the MPC8260ADS board, use an application called U-Boot, a ROM monitor and Linux boot loader developed specifically for custom boards that supports downloading and booting kernel images. U-Boot version 0.2.0 was used to download the Linux operating system described in this document. U-Boot can be downloaded at http://sourceforge.net/projects/u-boot. If another method for downloading the kernel is used, ignore the following section.

2.3.1 Compile and Execute U-Boot

To compile U-Boot, perform the following steps:

1. Unzip the downloaded U-Boot source file (u-boot-0.2.0.tar.bz2), if necessary, as follows:

   ```
   bunzip2 u-boot-0.2.0.tar.bz2
   ```

2. Extract the source code from the tar image using the tar command, as in the following example:

   ```
   tar -xvf u-boot-0.2.0.tar
   ```

   The options used with the tar command are as follows:

   — `x` Extract files from archive. If files are named on the command line, only those files are extracted from the archive. If several copies of a file exist in the archive, later copies overwrite earlier copies during extraction.

   — `v` Verbose operation mode.

   — `f` Filename where the archive is stored. Defaults to `/dev/rst0`.

3. Create a new directory (for example, `/usr/src/u-boot-0.2.0`) and move the extracted source files to it.
4. U-Boot is ready to be built. Drill down into the U-Boot directory by typing the following:

   ```
   cd /usr/src/u-boot-0.2.0
   ```

5. In the u-boot-0.2.0 directory modify the Makefile by adding the following line after the existing HOSTARCH definition:

   ```
   HOSTARCH:= ppc
   ```

6. In the Makefile, also comment out the CROSS_COMPILE= line (assuming that CROSS_COMPILE is defined in the environment variables). U-Boot is ready to be compiled.

   An important step to remember, especially between compilations, is the `make clean` command. Be sure to run the `make mrprope`**r** command to clear out old files. The `make clean` and `make mrproper` should not be required the first time U-Boot is built; rather, they should be used between future builds.

7. Before compilation, U-Boot must be configured for a specific board type. For the MPC8260ADS, type the following:

   ```
   make MPC8260ADS_config
   ```

8. To compile and create the U-Boot images, type the following:

   ```
   make all
   ```

When compilation is successful, `u-boot.srec`, `u-boot.map`, `u-boot.bin`, and a `u-boot elf` file are located in the main U-Boot directory. In this example, the `u-boot.srec` file is downloaded to the MPC8260ADS Flash memory.

2.3.2 Program U-Boot to Flash Memory

All development work for the example discussed here was performed using the Macraigor OCD Flash Programmer on a Windows NT environment. A SingleStep Blackbird COP interface was connected between the parallel port of the PC and the MPC8260ADS JTAG port connector. The Flash programmer should be set up with the following settings:

- Flash Type—LH28F016SCT
- Width—8 bits * 4 chips
- Starts at—0xFF800000
- Target RAM Starts at—0x04700000

The final step is to program `u-boot.srec` at 0xFFF00000. When this step is complete and the flash memory contents are verified, U-Boot should execute correctly from flash memory.

2.3.3 Test U-Boot Using Minicom

Minicom is a Linux text-based modem control and emulation program to communicate through the serial port to the target development board. It has excellent vt102 screen and keyboard emulation that includes full use of the entire cursor keypad and script capability. The script feature is particularly useful when automated login is required. To use Minicom, it must be configured. This step is easily achieved using the different menus available. Most problems with Minicom occur because the software is misconfigured or read or write permission errors occur during download to a target directory. On a correctly configured Debian system, Minicom can be installed by typing the following:

```
apt-get install minicom
```

Apt-get can compare the release levels of software on the host system with software located on a remote site. The system can be configured to update packages automatically when new releases become available. Minicom can also be downloaded from http://www.debian.org/distrib/packages. With the correct configuration setup for the target system/environment, the output of U-Boot can be viewed using Minicom. Complete the following steps to verify that U-Boot is executing correctly and to allow successful download of the kernel image discussed later in this document:

1. Start Minicom on the host machine and press CTRL + A then Z to enter the Minicom menu.
2. Raise the Comm Parameters menu and modify the settings to match the U-Boot build (for example, Bps/Par/Bits = 115200 8N1).
3. Raise the Configure Minicom menu.
   a) Select **FILENAMES AND PATHS**. Set the download and upload directories to `/tftpboot`.
   b) Select **FILE TRANSFER PROTOCOLS** and add an entry for tftp (for example, `tftp tftp/usr/bin/tftp YDNYN`).
   From left to right the seven options are the following:
   – `Name` Name that appears in the menu.
   – `Program` Path to the protocol
   – `Name` Defines whether the program needs an argument, for example, file to be transmitted.
   – U/D Defines whether this entry should appear in the upload or the download menu.
   – `Fullscr` Defines whether the program should run full screen.
   – `O-Red` Defines whether the minicom should attach the program standard input and output to the modem port.
   – `Multi` Can the protocol send multiple files with one command?
   c) Select **SERIAL PORT SET-UP**. Set up the serial device (for example, `/dev/ttyS0`).
   d) Select **SAVE SET-UP AS...** and enter a suitable filename (for example, MPC8260ADS).

A file called `minirc.8260ADS` is now located in the `/etc` directory of the host machine. When Minicom is required for the MPC8260ADS setup, typing `minicom 8260ADS` in the `/etc` directory can start it.

NOTE

Set the terminal emulation option to ANSI if the output of Minicom is to be captured in a text file.

When Minicom is configured, U-Boot can be tested. To do this testing, connect the serial port of the host machine to the terminal port (RS232-1) of the MPC8260ADS development board using a standard 9-way connector. With U-Boot programmed in flash memory when the board is turned on, an output similar to the following should be seen on Minicom:

```
U-Boot 0.2.0 (Feb 10 2003 - 16:35:34)
MPC8260 Reset Status: External Soft, External Hard
MPC8260 Clock Configuration

          Bus-to-Core Mult 3x, VCO Div 2, 60x Bus Freq 33-100, Core Freq 100-300
          dfbrg 0, corecnf 0x08, busdf 3, cpmdf 1, plldf 0, pllmf 1
          vco_out 266666664, scc_clk   66666666, brg_clk 66666666
          cpu_clk 199999998, cpm_clk 133333332, bus_clk 66666666

CPU:   MPC8260 (Rev 14, Mask unknown [immr=0x0062,k=0x002d]) at 199.999 MHz
Board: Freescale MPC8260ADS
I2C:   ready
DRAM: 16 MB
FLASH: 8 MB
In:    serial
Out:   serial
Err:   serial
Net:   FCC2 ETHERNET
Hit any key to stop autoboot: 0
=>
```

U-Boot commands can now be executed. To see a list of available commands, type `help`.

2.4 Download the Linux Kernel

The kernel source file can be downloaded from the Internet at http://www.kernel.org/.

2.4.1 Build a Linux Kernel Image

To build a typical Linux kernel, perform the following steps:

1. Unzip the downloaded the kernel source file (for example, linux-2.4.14.tar.bz2), if necessary, using the following bunzip2 command:

   ```
   bunzip2 linux-2.4.14.tar.bz2
   ```

2. The source code can be extracted from the tar image using the tar command. An example follows:

   ```
   tar -xvf linux-2.4.14.tar
   ```

3. Create a new directory (for example, `/usr/src/linux-2-4-14`) and copy the extracted source files into it.

   The new kernel is now ready to be built.

4. Drill down into the linux-2-4-14 directory by typing the following:

   ```
   cd /usr/src/linux-2-4-14
   ```

5. In the kernel Makefile comment out the CROSS_COMPILE= line. The Linux kernel is ready to be built.
6. Before compiling, type the following to ensure that any old `.o` files and dependencies are removed:
   ```
   make mrproper
   ```
7. To configure a new Linux kernel, make a configuration script to build the kernel at compile time. Type the following:
   ```
   make xconfig
   ```
   Configure the Linux kernel to be as large or small as required, depending on the number of drivers and support functions that are needed on the final application. At this stage, the true versatility of creating a Linux kernel becomes obvious because PCI, IP firewalls, IP masquerading, and many other options can be actively selected. When configuring the kernel prior to compilation, consider the target development board when selecting the kernel options. For example, building an unsupported function into the kernel when setting up the target for the MPC8260ADS board causes runtime errors.
8. The next stage in the build is to check and build the dependencies. Type the following:
   ```
   make dep
   ```
   The user-defined kernel is ready to be compiled. The options chosen during the “make xconfig” stage determine the applications and features that the kernel supports.
9. To make the binary image of the Linux kernel, type the following (noting the capital ‘I’):
   ```
   make zImage
   ```
   The compressed kernel image, `vmlinux.gz` should now be located in `/usr/src/linux-2-4-14/arch/ppc/boot/images`. This file is used to create the image that is downloaded to the target board. To create this file, use the mkimage command in the `/tools` directory of U-Boot, as the following example shows:
   ```
   ./mkimage -n ‘2.4.14 MPC8260ADS’ -A ppc -O linux -T kernel -C gzip -a 0 -e 0   -d
   /usr/src/linux-2-4-14/arch/ppc/boot/images/vmlinux.gz /tftpboot/8260image
   ```
   A downloadable image file, 8260image, is created in the /tftpboot directory.

The mkimage tool encapsulates the vmlinux image with header information for use with U-Boot. The options in the command line relate to the following features:

- –n Set image name
- –A Set architecture (for example, ppc)
- –O Set operating system (for example, Linux)
- –T Set image type (for example, ramdisk)
- –C Set compression type (for example, gzip)
- –a Set load address
- –e Set entry point
- –d Point to location of image data

The next section discusses how to run the image on the MPC8260ADS.

2.4.2 Download the Kernel onto the MPC8260ADS

Take the following steps to download the kernel to the MPC8260ADS.

1. Start Minicom and power-on the MPC8260ADS.

   U-Boot should start up.
2. At the prompt, type `printenv`.

   This command lists the environment variables for the system.
3. Use the setenv command (for example, `setenv ipaddr 192.168.1.52`) to set up the following variables (the following are example values):
   - baudrate = 15200
   - serverip = 192.168.1.1
   - ipaddr = 192.168.1.52
   - bootfile = `/tftpboot/8260image`

   Type `printenv` to confirm that the settings are correct.
4. Ensure that the ethaddr variable is also configured with an appropriate MAC address for the target board.
5. Use the setenv command to configure the bootcmd variable to run the tftpboot and bootm commands. Type the following command in U-Boot:

   ```
   setenv bootcmd tftpboot \; bootm
   ```

   When tftpboot runs, the board sends out tftp requests to the PC to download the kernel image. Bootm then executes and runs the kernel image from its location in memory (for example, 0x100000).
6. Type `boot` to execute the commands in the bootcmd variable.

The kernel is now running on the target board and attempts to mount a root file system. Decide how to set up the system to accomplish this task. An example setup is shown in the next section.

2.5 Mount the Root File System

You must now decide how the file system is to be mounted. The example given here is a point-to-point system in which the Linux machine/partition used for kernel download also acts as a network file system (NFS). To implement this system, the U-Boot variable bootargs must be initialized with the following parameters before the kernel is downloaded:

```
bootargs = root = /dev/nfs rw nfsroot = <HOST IP ADDRESS>:<DIRECTORY OF TARGET FILE
SYSTEM> nfsaddrs = <TARGET IP ADDRESS>:<HOST IP ADDRESS>:<GATEWAY IP
ADDRESS>:<NETMASK> init = <LOCATION OF INITIAL APPLICATION>
```

The following example shows how to initialize bootargs in U-Boot:

```
setenv bootargs root=/dev/nfs rw nfsroot=192.168.1.1:/tftpboot/192.168.1.52
nfsaddrs=192.168.1.52: 192.168.1.1:192.168.13.254:255.255.255.0 init=/bin/sh
```

Now when `boot` is typed and the kernel is downloaded and executed, it mounts a root file system and runs sh from the bin directory, assuming that sh is compiled for PPC and placed in the correct location.

It is important to put an entry for the target board in the `/etc/exports` file, as follows:

```
/tftpboot/192.168.1.52                  192.168.1.52 (rw, no_root_squash)
```

When the exports file is set up correctly, type the following in the `/usr/sbin` directory:

```
./exportfs -va
```

When the NFS is in place, the request to mount the root file system is fulfilled and applications can execute on the system.

2.6 Build Applications

When the kernel is complete and the root file system is configured, applications can be built to run on the MPC8260ADS board. These applications can range from telnet to the apache web server. The development tools containing the PPC cross compiler usually contain application source files that can be built for a PPC target. A couple of these software packages are discussed here. Note that when the applications are built, the system must be configured to run them correctly.

2.6.1 BusyBox

BusyBox provides basic utilities for embedded systems that are ideal as minimalist replacements for most common utilities on the desktop system (for example, sh, tar, ls, and so on). It is a very good starting-point for developing an effective and efficient embedded Linux file system. Following is an example of building BusyBox (v0.60.5) for an MPC8260 system:

1. Download the BusyBox source code from http://www.busybox.net/.
2. Extract the BusyBox files using the tar command.
3. Modify the `Config.h` file to define or undefined the applications to make during the build.
4. If necessary change the CROSS field in the main BusyBox Makefile to point to the PPC cross compiler.
5. Type `make clean`.
6. Type `make install`.
7. The utilities for the system are now built and can be copied from the _install directory of BusyBox to the location of the root file system to be mounted on the target board (for example, `/tftpboot/192.168.1.52`).

2.6.2 Boa Web Server

A computer that delivers web pages is a *web server*. Every web server has an IP address and possibly a domain name. For example, entering the URL `http://www.pcwebopedia.com/index.html` in your browser sends a request to the server whose domain name is pcwebopedia.com. The server then fetches the page named index.html and sends it to your browser. Installing server software and connecting the machine to the Internet can turn any computer into a Web server. Boa is a single-tasking HTTP server. Unlike traditional web servers, it does not fork for each incoming connection, and it does not fork many copies of itself to handle multiple connections. Boa internally multiplexes all of the ongoing HTTP connections and forks only for common gateway interface (CGI) programs, which must be separate processes.

The following steps show how to build Boa for an MPC8260ADS system:

1. Download the Boa source code from http://www.boa.org/.
2. Extract the Boa files using the tar command.
3. Type `make clean`.
4. Type `./configure`.
5. If necessary, change the CC field in the main Boa Makefile to point to the PPC cross compiler.
6. Type `make all`.

A Boa executable file should be located in the main Boa directory. Copy this file and relevant other files to the root file system for the target.

3 Test Linux on the MPC8260ADS

The Linux version used on the host was Debian Linux r3.0. The serial port (COM1) was configured to operate at 115200 baud rate with no parity or stop bits and 8 data bits using the Minicom application. The serial port of the host Linux PC was connected to the terminal port (RS232-1) of the MPC8260ADS development board using a standard 9-way connector. The exact details of the boot-up sequence vary, depending on the applications that were selected during the kernel configuration stage. The following output shows an example of the U-Boot starting up and the kernel being downloaded and executed.

```
U-Boot 0.2.0 (Feb 10 2003 - 16:35:34)
MPC8260 Reset Status: External Soft, External Hard
MPC8260 Clock Configuration
     - Bus-to-Core Mult 3x, VCO Div 2, 60x Bus Freq 33-100, Core Freq 100-300
     - dfbrg 0, corecnf 0x08, busdf 3, cpmdf 1, plldf 0, pllmf 1
     - vco_out 266666664, scc_clk   66666666, brg_clk 66666666
     - cpu_clk 199999998, cpm_clk 133333332, bus_clk 66666666
CPU:   MPC8260 (Rev 14, Mask unknown [immr=0x0062,k=0x002d]) at 199.999 MHz
Board: Freescale MPC8260ADS
I2C:   ready
DRAM: 16 MB
FLASH: 8 MB
In:    serial
Out:   serial
Err:   serial
Net:   FCC2 ETHERNET
Hit any key to stop autoboot: 0
=> boot
ARP broadcast 1
TFTP from server 192.168.1.1; our IP address is 192.168.1.52
Filename '/tftpboot/8260image'.
Load address: 0x100000
Loading: *#################################################################
     ################################################
done
Bytes transferred = 577324 (8cf2c hex)
## Booting image at 00100000...
Image Name:   31/01/03 uboot SF test
Image Type:   PowerPC Linux Kernel Image (gzip compressed)
Data Size:    577260 Bytes = 563.7 Kbytes
```

```
Load Address: 00000000

Entry Point: 00000000

Verifying Checksum... OK

Uncompressing Kernel Image... OK

Linux version 2.4.14 (root@MJOHNSTON) (gcc version 2.95.3 20010315
(release/MontaVista)) #2 Fri Jan 31 10:48:34 GMT 2003

On node 0 totalpages: 4096

zone(0): 4096 pages.

zone(1): 0 pages.

zone(2): 0 pages.

Kernel command line: root=/dev/nfs rw nfsroot=192.168.1.1:/tftpboot/192.168.1.52
nfsaddrs=192.168.1.52:192.168.1.1:192.168.13.254:255.255.255.0 init=/bin/sh

Warning: real time clock seems stuck!

Calibrating delay loop... 133.12 BogoMIPS

Memory: 14588k available (1068k kernel code, 388k data, 56k init, 0k highmem)

Dentry-cache hash table entries: 2048 (order: 2, 16384 bytes)

Inode-cache hash table entries: 1024 (order: 1, 8192 bytes)

Mount-cache hash table entries: 512 (order: 0, 4096 bytes)

Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)

Page-cache hash table entries: 4096 (order: 2, 16384 bytes)

POSIX conformance testing by UNIFIX

Linux NET4.0 for Linux 2.4

Based upon Swansea University Computer Society NET3.039

Starting kswapd

devfs: v0.120 (20011103) Richard Gooch (rgooch@atnf.csiro.au)

devfs: boot_options: 0x0

CPM UART driver version 0.01

ttyS00 at 0x8000 is a SCC

pty: 256 Unix98 ptys configured

block: 64 slots per queue, batch=16

RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize

fec: Phy @ 0x0, type 0xe040000f

fec: Phy @ 0x1, type 0x00000000

fec: Phy @ 0x2, type 0x00000000

eth0: FCC ENET Version 0.2, 08:00:22:50:70:63

 (0 Mbit, Half Duplex)

loop: loaded (max 8 devices)

PPP generic driver version 2.4.1

NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
```

```
TCP: Hash tables configured (established 1024 bind 1024)
IPv4 over IPv4 tunneling driver
IP-Config: Gateway not on directly connected network.
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
Looking up port of RPC 100003/2 on 192.168.1.1
Looking up port of RPC 100005/1 on 192.168.1.1
VFS: Mounted root (nfs filesystem).
Freeing unused kernel memory: 56k init
init started: BusyBox v0.60.5 (2002.11.20-15:23+0000) multi-call
Starting internet superserver: inetd.

Please press Enter to activate this console.
BusyBox v0.60.5 (2002.11.20-15:23+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

sh: can't access tty; job control turned off
#
```

A Linux kernel is now running on the MPC8260ADS. From the command line on Minicom applications located on the root file system can be run.

4 Data Sources & References

1. The Future of Linux, Ruediger Berlich (22.11.2000)
   http://www.linux-knowledge-portal.org/en/content.php?&content/future/linux_fut.html.
2. How Hard is Real-time?, Jeff Dionne (16.09.2000)
   http://www.linuxdevices.com/articles/AT3524337625.html
3. Cross Compiling
   http://www.cygwin.com/xfree/docs/cg/prog-build-cross.html

NOTE

Freescale does not endorse any software supplier or developer mentioned in this document. Freescale also does not accept any responsibility for the reliability or support of the code presented in this text. The example discussed in this document is purely to illustrate how Linux can be embedded onto the MPC8260ADS. The code discussed is GNU open source. All software restrictions and rules apply.

5 Revision History

Table 1 shows the revision history of this document.

Table 1. Revision History

Revision Number	Changes
0.0	Initial release
0.1	Nontechnical reformatting
1	Nontechnical reformatting and editing.

THIS PAGE INTENTIONALLY LEFT BLANK

Document Number: AN2579
Rev. 1
11/2006