Linux on the Innovator

(Revision 1.0)

DELPHI
Communication Systems, Inc.
www.delcomsys.com

25 February, 2003

Revision	Date	By	Comment
1.0	2003-02-25	wpd	Initial Release

Table of Contents

1. Introduction

So, you've gotten one of the Innovator platforms with the exciting TI OMAP processor on it and you want to run Linux on the ARM processor. Should be simple right? Just turn it on! Well, maybe not. It didn't ship with Linux on it. This document will discuss the steps we took to get Linux up and running on our Innovator. We hope others will find it useful in getting similar results. Feel free to send email to `info@delcomsys.com` with any questions or comments you have regarding the material in this document.

This document is divided into two main sections: **Binary Installation** and **Source Installation**. Developers who just want to get the Linux kernel up and running on the OMAP, should read (and follow the directions given in) the **Binary Installation** section. Those who are interested in compiling everything from source code should read both sections.

First, lets look at an overview of what we want to accomplish. The end goal is that we want to be able to run Linux on the Innovator. In order to do this, we need some way to load it on the board, ideally through the ethernet port on the breakout board (which is significantly faster than the serial port, and even faster than plugging the bits in one-by-one by hand). We will also need a root file system and a set of applications to be run by the kernel. This document presents the technique of loading an initial ramdisk from memory. Other options include storing it in flash memory or mounting it via NFS.

In order to reach your goal of running Linux on your Innovator, you will need a certain amount of hardware, software, and TCP/IP information, as described below:

Hardware

- Innovator with breakout board, ethernet cable, serial cable adapter, and a null-modem serial cable.
- Emulator. We use the Spectrum Digital SPI515 parallel port emulator.
- A Linux workstation
- A PC running windows. We used Windows 98 running under Win4Lin on our Linux development box – see Linux and Windows – Cohabitation.
- A JTAG Debugger to use with the TI development tools. We used a Spectrum Digital SPI515 parallel port JTAG debugger.

Software

- Linux. We have used Red Hat 7.2[1] and 8.0.
- Windows
- Code Composer Studio for the OMAP (referred to as CCS throughout this document). We used version 2.0.

1 The x86 binaries for the toolchains that are installed in Section 2 were all compiled with the RedHat 7.2 machine, so that they will run on both RedHat 7.2 and RedHat 8.0 machines. (In other words, they are linked against the libraries that were shipped with RH 7.2 rather than the libraries that were shipped with RH 8.0.)

- Terminal emulator software. Throughout this document, we will assume that the terminal emulator software you use is `minicom`, which runs on the Linux workstation, although you could use the HyperTerminal program installed on your Windows PC if you desire.
- A tftp server. We use the default one that shipped with RedHat.
- Linux on the Innovator files. The latest version of this document, and all of the open source files referenced therein are available at ftp://ftp.delcomsys.com/pub/omap/loti. Everything that you need for Section 2, **Binary Installation**, is available in the `loti.tar.bz2` file. All of the source distributions referenced in Section 3, **Source Installation**, are available on the Delphi FTP site for your one stop shopping convenience.

TCP/IP Information

- IP address of your Linux workstation (`default server IP address`). This may be dynamically assigned via DHCP, or statically assigned by your MIS department. You can use the `/sbin/ifconfig` command to display the IP address of your workstation if you are unsure of what it is.
- IP address information for your Innovator. This must be provided to you by your MIS department and should include the items listed below. In particular, the IP address for your Innovator should be a static IP address. (RedBoot does support the BOOTP and DHCP protocols, but life is much simpler[2] if you dedicate an IP address to your Innovator board). The terms in parentheses will be used later when configuring the RedBoot bootloader.
  - gateway/router address (`Gateway IP address`)
  - IP address (`Local IP address`)
  - netmask (`Local IP address mask`)
  - DNS server IP address (`DNS server IP address`)

Side Note: Linux and Windows - Cohabitation

As you get into putting Linux on the ARM side of the Innovator, you will find that you are pulled more and more into using tools that run under some variant of the Linux operating system - at least if you follow this document you will. However, the TI Code Composer Studio (CCS) tools do not run natively under Linux. So, you will find that you need a Windows box running Code Composer to build code that runs on the DSP or to perform JTAG debugging under CCS. We use the gcc tools for code that runs on the ARM9 core of the OMAP. We found it natural to use Linux versions of those tools. So, we needed another PC that ran Linux for this work.

2 Well, life is simpler for us, since we don't need to try to document how to set up a BOOTP server on your network. It's also simpler for you, since you don't need to figure out how to set up a BOOTP server on your network.

Does all of this seem frustrating? It can be. We would sure love it if the TI tools ran under Linux. We did indeed get frustrated with the need to have two boxes (Windows and Linux). Skipping all the in-between steps (like dual boot), we settled on a single PC setup. The PC runs RedHat Linux with Win4Lin (available at http://www.netraverse.com) running a Windows 98 installation in a separate window on the Linux desktop. The great thing about the Win4Lin approach was that the Code Composer Tools ran fine and Win4Lin, Linux, and the parallel port handled the parallel port emulator without a hitch.

So, with the previously described setup, we could do all of the necessary development from one computer without any need to reboot to switch operating systems. It could be important. You may be thinking, "When I am done, I will be doing all of my development using GPL tools under Linux and I won't need Windows anymore." We like to remember the oft forgotten part of the OMAP, the 55x DSP. We're going to want to develop code to run there too and we will need the TI tools to do that!

2. Binary Installation

Install the Host Based Software

In order to run Linux on the Innovator, you will need to install some software on your Linux and Windows workstation(s).

- Install the Code Composer Studio tools on a Windows PC (or in a Win4Lin window on your Linux workstation) according to the instructions included with the Innovator.
- Install your favorite tftp server. If you use a RedHat workstation, you should be able to install the tftp server RPM with a command similar to:

  ```
  prompt# rpm -ivh /mnt/cdrom/RedHat/RPMS/tftp-server-0.17-14.i386.rpm
  ```

  Consult the documentation that came with your Linux distribution for more details. Note that once you install the tftp server you must arrange for it to be enabled by editing the appropriate configuration file, most likely `/etc/xinetd.d/tftp` or possibly `/etc/inetd.conf`. You will also have to create a root directory for tftp transfers if it was not created with the `rpm` command. We suggest that you use `/tftpboot`. (Whatever you choose, it must match the directory given with the `-s` option in the configuration file listed above.) You will have to create the directory as root, but we suggest that you make it writable from your normal user account. (There are some operations that must be performed as root when dealing with embedded Linux development, but simply copying files to a directory so that they may be fetched over the network is not one of them.) Finally, you may have to (re)configure the firewall settings on your workstation to allow TFTP transfers to take place. Instructions for configuring your firewall settings are beyond the scope of this document, but if you think that everything is set up properly, but you still cannot transfer files from your workstation using TFTP, consider disabling your firewall (for the duration of a simple test) using a command similar to the following:

```
prompt# /etc/init.d/iptables stop
```

  or

```
prompt# /etc/init.d/ipchains stop
```

  If that solves your problem, then reconfigure your firewall settings to allow UDP port 69 through, or convince somebody more knowledgeable about Linux and firewall settings to do that for you.

- Make sure that you have write access to `/usr/local/xtools`. We have found that the simplest way to do this is to create a group[3] named `xtools`; add yourself and whomever else you want to be able to write to the `/usr/local/xtools` directory

3 If you are unfamiliar with the concept of groups in Linux, consult your documentation, or any of the books available about Linux system administration. You may also find it simpler to run the GUI tool provided by RedHat for user group management.

to that group; change the group ownership of `/usr/local/xtools` to be `xtools`; and make it group writable and "set-group-id" (sgid). This is summarized in the example below:

Edit `/etc/groups` as root and add a new group called `xtools` with whomever you want in the group. Then...

```
prompt# mkdir /usr/local/xtools
prompt# chgrp xtools /usr/local/xtools
prompt# chmod g+ws /usr/local/xtools
```

- Now we get to the fun stuff: `loti.tar.bz2`. This file contains the RedBoot bootloader, the binary image of the Linux kernel, the binary image of the initial root file system, and the toolchain required to build ARM Linux executables. Unpacking and installing this file requires two steps. First create a working directory. Throughout this document, we assume that `loti.tar.bz2` was unpacked in a directory named `~/omaplinux`. Unpack the tarball. Then change to the `/usr/local/xtools` directory and unpack the toolchain. The example below shows these steps. It is assumed that `loti.tar.bz2` was downloaded to a directory named `/dist/omap` on the local workstation.

```
prompt$ mkdir ~/omaplinux
prompt$ cd ~/omaplinux
prompt$ tar xjf /dist/omap/loti.tar.bz2
prompt$ cd /usr/local/xtools
prompt$ tar xjf ~/omaplinux/toolchain.tar.bz2
```

At the completion of these steps, you should have the following files installed in `~/omaplinux`:

```
hello.c  initrd      mkrd               redboot-sram.out  toolchain.tar.bz2
Image    initrd.dir  redboot-flash.bin  rootfs
```

Don't forget to add `/usr/local/xtools/arm-uclibc-3.2.1/bin` to your path using a command similar to the one shown below (if you use bash):

```
prompt$ export PATH=/usr/local/xtools/arm-uclibc-3.2.1/bin:$PATH
```

or, if you prefer csh variants:

```
prompt$ setenv PATH /usr/local/xtools/arm-uclibc-3.2.1/bin:$PATH
```

Most people add this in their `.bashrc` or `.cshrc` file so that it is always available.

Install RedBoot

The Linux kernel is not self-booting. It relies on a bootloader to establish a stable system state, to gather system information, to feed it to the kernel, and to start (boot) the kernel. Since we have successfully used the eCos real time kernel (RTK) in other projects, and since it includes a very full-featured bootloader, (called RedBoot), we decided to port RedBoot to the OMAP. Also, we felt confident that we could get ethernet support running under RedBoot fairly quickly so that we could download code via ethernet instead of the serial port. (As an aside, if you are looking for a very configurable, low-profile, open-source RTK for an embedded OMAP application you might want to check out eCos at http://sources.redhat.com/ecos. It could be the OS for you.)

One of the files that you unpacked from `loti.tar.bz2` was `redboot-sram.out.` This file contains a limited version of the RedBoot bootloader that we shall use to install the full version in FLASH. In order to run this program, you must have a working version of Code Composer Studio. You will also need a serial port terminal program, such as `minicom`.

All of the instructions we have seen for installing software on the Innovator start off by stating that you should set the DIP switches in a certain manner. So, we shall follow suit. We have found that if we leave the DIP switches in their default factory configuration (DIP switch 1 on and switches 2, 3, and 4 off), then we never have to think of them again. (This maps the boot flash to chip select 0, or, equivalently, address 0. Since the code in RedBoot assumes that the boot flash is at physical address 0, and makes no assumptions about the user flash devices, this seems to be the safest way to go). Depending on what is installed in the boot flash, Code Composer may not operate cleanly. The most common problem we have seen is that CCS is unable to access the DSP from its startup GEL script. Since we are not concerned with the DSP at this point of the installation, you can safely ignore any warnings or error messages regarding the DSP. The second most common problem we have seen is that CCS is unable to disassemble the instructions in memory correctly. Our current theory is that the MMU is not fully supported by CCS and that it fetches instructions and data based on the physical address rather than the virtual address given. Regardless, if you follow the instructions given below, this should not be an issue.

- Install the emulator, ensure that jumpers JP1 and JP2 are on pins 2-3 so that the TI emulator header is selected.
- Connect the serial port cable to the Interface Module.
- Connect a null-modem cable between the (Innovator) serial port connector labeled “COM1” and your PC running `minicom`.
- Configure `minicom` for 115200 baud, 8 bits, no parity.
- Apply power to the board. Turn the switch on if you need to. (We have found that the simplest way to turn the board on and off is to plug the power supply “brick” into a power strip dedicated to this purpose and to simply turn the power strip on and off.)
- Start CCS (we assume you have already run Code Composer Setup and have setup for your emulator and for the Innovator). As we stated earlier, you may safely ignore any warnings regarding the inability of CCS to communicate with the DSP. You may not ignore warnings regarding the inability of CCS to communicate with the ARM processor.
- Open a debug window for the ARM processor.

- Select “Tools|TMS470R2x Advanced Features...”
- Check the box marked “Break on Reset”
- Press the F5 key (some people do this by selecting “Debug|Run”). You will be presented with a dialog box that warns you that you have not loaded any code on the processor yet and which asks you if you want to continue. Select “Yes”.
- Press the RESET button on the Innovator. (By the way, if you missed this in the manual, if you press and hold the RESET button for 2 seconds or longer, it generates a full Power On reset. If you just tap it, it generates an ARM only reset – either reset is fine here).
- At this point, the Disassembly window should show that the PC is at its reset vector address of 0. This sequence (Start CCS, check the “Break on Reset” box, run the processor, press the reset button) is a handy technique for getting the processor into a consistent state with which CCS is happy. And, we didn't have to muck with any DIP switches!
- Load the `redboot-sram.out` program on the processor. You will receive some error messages from CCS stating that it is unable to load memory at address 0x20, etc... You may safely ignore any of these messages as long as the address is less than 0x20000000. Someday, we might figure out a way to generate an ELF file with the GNU tools that CCS is happy to load without any problems. Unfortunately, today is not that day.
- Run the application you just loaded (i.e. press F5 or select “Debug|Run”). You should see a output similar to the following in your `minicom` window:

```
+FLASH configuration checksum error or invalid key

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 13:07:29, Jan 28 2003

Platform: Innovator (ARM9)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x0000e238-0x01fed000 available
FLASH: 0x10000000 - 0x10400000, 64 blocks of 0x00010000 bytes each.
RedBoot>
```

- The final line is the “RedBoot” prompt. The examples that follow all start with such a prompt.
- Initialize the “flash image system”. This is a file system stored on the flash device that may be manipulated with RedBoot.

```
RedBoot> fis init -f
```

You will see a display similar to the following:

```
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
... Erase from 0x10020000-0x103e0000:
............................................................
... Erase from 0x103f0000-0x103f0000:
... Erase from 0x10400000-0x10400000:
... Erase from 0x103f0000-0x10400000: .
... Program from 0x01fef000-0x01fff000 at 0x103f0000: .
RedBoot>
```

- Load the ROM (i.e. flash) version of RedBoot using the ymodem protocol. This is probably a good time to tell you that you should have started `minicom` in the same directory as you unpacked the files from the `loti.tar.bz2` tarball (i.e `~/omaplinux`). If you did that, then when you attempt to send the `redboot-flash.bin` file to the Innovator, you won't have to search very far for it. At the `RedBoot>` prompt, enter the following:

```
RedBoot> load -r -b 0x80000 -m ymodem
```

This commands RedBoot to load a raw (`-r`) file into memory starting at a base address (`-b`) of `0x80000`[4] and to use the Y-Modem protocol to receive the file via the serial port. You will need to command `minicom` to download the file by typing Control-a S, selecting ymodem, and scrolling down to the `redboot-flash.bin` file, pressing the space bar to select it, and pressing enter to start the download. When the download completes, you will be presented with a `RedBoot>` prompt once again and you should see output similar to the following:

```
Raw file loaded 0x00080000-0x0009f5eb, assumed entry at 0x00080000
xyzModem - CRC mode, 1007(SOH)/0(STX)/0(CAN) packets, 4 retries
RedBoot>
```

4 Earlier we stated that by configuring the DIP switches the way we did, that flash would be at address `0`. It is tempting to think that this command will download the image directly to flash starting at address `0x80000`. Unfortunately, this is not the case. eCos, and hence, RedBoot, requires that RAM be at address 0 so that it can overwrite the interrupt vector table. RedBoot, and hence eCos, uses the MMU in the ARM to swap the addresses of flash and SDRAM. Thus, flash occupies virtual memory from address `0x10000000` to `0x10400000` while SDRAM occupies virtual memory from address `0x00000000` to `0x02000000`.

- At this point, you have successfully transferred the RedBoot image from your host computer to RAM on the Innovator. Now it is time to write it to the flash using the commands shown below:

```
RedBoot> fis write -f 0x10000000 -b 0x80000 -l 0x30000
* CAUTION * about to program FLASH
            at 0x10000000..0x1002ffff from 0x00080000 - continue (y/n)? y
... Erase from 0x10000000-0x10030000: ...
... Program from 0x00080000-0x000b0000 at 0x10000000: ...
RedBoot>
```

This commands RedBoot to write to the flash starting at address `0x10000000` from memory starting at address `0x80000` and to write `0x30000` bytes. Since you are mucking about with the flash, RedBoot kindly asks you if you are out of your mind or not.

- Now you have erased the iBoot bootloader (or whatever else was stored in the Boot flash) and replaced it with RedBoot. Exit CCS, turn off the power to the board, unplug the emulator, (or unplug it while it's hot, if you dare), and turn the board back on. After a little while, you should see the following on the serial terminal:

```
+FLASH configuration checksum error or invalid key
no link
no link
no link
no link
Ethernet eth0: MAC address 00:0b:36:00:01:74
Can't get BOOTP info for device!

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 09:12:44, Jan 27 2003

Platform: Innovator (ARM9)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x00015c18-0x01fe1000 available
FLASH: 0x10000000 - 0x10400000, 64 blocks of 0x00010000 bytes each.
RedBoot>
```

Don't be alarmed by the long pause at the beginning of the boot sequence. By default, RedBoot attempts to determine its IP settings using the BOOTP protocol. Once that times out, RedBoot continues with its boot sequence.

- Now it is time to tell RedBoot about all of those IP parameters we told you to obtain way back at the beginning of this document. We do this with the `fconfig` command as shown in the example below. Remember to supply your own parameters appropriate to your network settings.

```
RedBoot> fconfig
Run script at boot: false
Use BOOTP for network configuration: false
Gateway IP address: 10.1.0.100
Local IP address: 10.1.3.2
Local IP address mask: 255.255.0.0
Default server IP address: 10.1.0.201
DNS server IP address: 63.110.40.25
GDB connection port: 9000
Force console for special debug messages: false
Network hardware address [MAC]: 0x00:0x00:0x00:0x00:0x84:0xDF
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)? y
... Erase from 0x103e0000-0x103e1000: .
... Program from 0x01fe2000-0x01fe3000 at 0x103e0000: .
RedBoot>
```

Now that you have configured the IP settings, if you press the reset button, or power cycle the board, you should see:

```
+Ethernet eth0: MAC address 00:0b:36:00:01:74
IP: 10.1.3.2/255.255.0.0, Gateway: 10.1.0.100
Default server: 10.1.0.201, DNS server IP: 63.110.40.25

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 09:12:44, Jan 27 2003

Platform: Innovator (ARM9)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x00015c18-0x01fe1000 available
FLASH: 0x10000000 - 0x10400000, 64 blocks of 0x00010000 bytes each.
RedBoot>
```

Congratulations! You now have an Innovator capable of loading and running arbitrary programs from the flash, the ethernet, or the serial port. Read on to see how to load one of those arbitrary programs (namely, Linux).

Use RedBoot to load the kernel and the initial ramdisk

In this section, we present instructions for downloading the kernel and the initial ramdisk and writing them to the flash. In the next section, we will present instructions for loading the initial ramdisk and the kernel from flash for execution. It should be noted that it is not required that you first write the component (i.e. the kernel or the initial ramdisk) to the flash before executing the kernel. During development, it is often easiest to keep one component in the flash while downloading the other one.

In order to perform the steps described in this section, you must first ensure that you have a tftp server running, presumably on your Linux host. Instructions for doing this were given previously in this document. If you skipped those instructions, go back and reread them now. Once you have created the `/tftpboot` directory and made it writable by you, copy the `Image` and `initrd` files from your `~/omaplinux` directory to `/tftpboot`. Once you have done all of that, follow the steps below to load the Linux kernel (which is in the file named `Image`) and the initial ramdisk (which is in the file named `initrd`) into flash.

- Download the kernel using a command similar to the one shown below. Note that if you set the "Default Server IP Address" parameter in fconfig to the IP address of your workstation running the TFTP server, you may omit the `-h <<server IP address>>` option below. This command loads a raw (`-r`) binary file named `Image` into memory at a base address of `0x80000` from a tftp server at address `<<server IP address>>`. While it is downloading, the cursor will display a "spinning helicopter" pattern (`-v`).

```
RedBoot> load -v -r -b 0x80000 -h <<server IP address>> Image
/
Raw file loaded 0x00080000-0x001bccbf, assumed entry at 0x00080000
RedBoot>
```

- Write the kernel to flash using a command similar to the one shown below. This creates a new file in the flash image system named, oddly enough, `Image`. Note that if a file already exists by that name, you will be questioned as to whether or not you want to overwrite it. Also note that, if you choose to overwrite the existing file with a new one, the new one must be small enough to fit in the space allocated for the existing one. If that is not the case (and you will be informed if it is not the case), simply delete the existing file (using the `fis delete` command) and try again.

```
RedBoot> fis create Image
... Erase from 0x10030000-0x10170000: ....................
... Program from 0x00080000-0x001bccc0 at 0x10030000: ....................
... Erase from 0x103f0000-0x10400000: .
... Program from 0x01fef000-0x01fff000 at 0x103f0000: .
RedBoot>
```

- Download the initial ramdisk using a command similar to the one shown below.

```
RedBoot> load -v -r -b 0x400000 -h <<server IP address>> initrd
/
Raw file loaded 0x00400000-0x0042de8b, assumed entry at 0x00400000
RedBoot>
```

- Write the initial ramdisk to flash using a command similar to the one shown below.

```
RedBoot> fis create initrd
... Erase from 0x10170000-0x101a0000: ...
... Program from 0x00400000-0x0042de8c at 0x10170000: ...
... Erase from 0x103f0000-0x10400000: .
... Program from 0x01fef000-0x01fff000 at 0x103f0000: .
RedBoot>
```

Boot the kernel

At this point, you have RedBoot, a Linux kernel, and an initial ramdisk stored in the flash (along with some configuration parameters used by RedBoot). We are finally ready to try running our Linux kernel. This is surprisingly easy, as the next set of commands show:

```
RedBoot> fis load initrd
RedBoot> fis load Image
RedBoot> exec -r 0x400000
Using base address 0x00080000 and length 0x00140a1c
Linux version 2.4.19-rmk4 (wpd@akula) (gcc version 3.2.1) #1 Fri Jan 31 09:55:18
 EST 2003
CPU: ARM/TI Arm925Tsid(wb) revision 2
Machine: Innovator/OMAP1510
```

... (output deleted)

The first command copies the initial ramdisk from flash into RAM (starting at the address where it was originally downloaded, namely `0x400000`). The second command copies the kernel from flash into RAM (at address `0x80000`). The final command starts execution of the kernel and tells it to find its initial ramdisk at address `0x400000`. Note that the order of the two "load" commands is important. You must load the initial ramdisk image prior to loading the kernel (because the `exec` command assumes that the kernel was the last file loaded). As mentioned previously, you can load either (or both) the initial ramdisk or the kernel via tftp from your host computer or, as shown above, from the flash. Typically, when we are developing applications to run in Linux, we keep the kernel in flash and simply reload the initial ramdisk via tftp each time we reboot the Innovator[5]. Other options, such as mounting the root filesystem via NFS, or using a ROM filesystem or a compressed RAM filesystem are beyond the scope of this document.

If everything worked properly, you should see messages similar to the following:

```
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 4096K
VFS: Mounted root (ext2 filesystem).
Freeing init memory: 72K
init started:  BusyBox v0.60.5 (2003.01.17-14:50+0000) multi-call binary
Starting Network

Please press Enter to activate this console.
```

Press Enter and you should see the following:

```
BusyBox v0.60.5 (2003.01.17-22:02+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

#
```

Now you are ready to initialize your TCP/IP parameters for the Linux kernel.

5 You can even use the `fconfig` RedBoot command to specify a startup script that loads one image via TFTP and the other from the flash and then runs the kernel. That way you can simply reboot the Innovator to get the latest version of which ever image you are playing with on a given day.

Initialize TCP/IP

The initial ramdisk included in `loti.tar.bz2` does not enable the network for the simple reason that we have no idea what your network parameters are. So you must manually enter three commands to initialize the network. Enter commands similar to the ones shown below, but substitute the appropriate IP address, netmask, broadcast address[6], gateway address, and name server address for the 10.1.3.2, 255.255.0.0, 10.1.255.255, 10.1.0.100, and 63.110.40.25 values shown below. Note that these parameters are identical to the ones you used for your RedBoot bootloader.

```
# ifconfig eth0 10.1.3.2 netmask 255.255.0.0 broadcast 10.1.255.255
# route add default gw 10.1.0.100 eth0
# echo nameserver 63.110.40.25 > /etc/resolv.conf
```

The first command configures the ethernet interface named `eth0` (which is the only ethernet interface on the Innovator) with an IP address of `10.1.3.2`, a netmask of `255.255.0.0`, and a broadcast address of `10.1.255.255`. The second command specifies the default gateway (`10.1.0.100`) to be used to route packets destined for hosts that are not on the local subnet. This is not required if you never intend to communicate with hosts outside of your local subnet. The third command specifies the IP address of the DNS nameserver (`63.110.40.25`) which is used to resolve textual host names such as `www.ti.com` into raw IP addresses such as `192.91.75.198`. Once again, this is not required if you never intend to resolve textual host names.

If you want to verify that the network is set up properly, you can use the `ping` command as shown below.

```
# ping www.ti.com
PING www.ti.com (192.91.75.198): 56 data bytes
64 bytes from 192.91.75.198: icmp_seq=0 ttl=237 time=70.0 ms
64 bytes from 192.91.75.198: icmp_seq=1 ttl=237 time=70.0 ms
64 bytes from 192.91.75.198: icmp_seq=2 ttl=237 time=70.0 ms
^C
--- www.ti.com ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 70.0/70.0/70.0 ms
#
```

Compile and run the“Hello World” program.

Now that you have Linux running on your Innovator, you probably want to start running your own Linux application. Instructions for writing your own application for your new multimedia enhanced, wireless, USB capable, ultra low power, lemony scented super duper application are beyond the scope of this document, but we can help you get started by showing you how to compile and run the traditional “Hello World” program. We'll start by showing you the program:

6 The broadcast address may be obtained by ANDing the netmask with your IP address and replacing the least significant “0” bits with “1” bits.

```
#include <stdio.h>

int
main(int argc, char *argv[])
{
  printf("Hello World\n");
  return(0);
}
```

If you are particularly lazy, and don't want to type this in yourself, you can find this program in the `hello.c` file in the `omaplinux` directory. Compile this program on your Linux workstation using the `arm-uclibc-gcc` compiler you installed way back at the beginning of this document:

```
prompt$ arm-uclibc-gcc -static -o hello hello.c
prompt$ cp hello /tftpboot
```

The first command compiles and statically links the program. By default, gcc will generate an executable that uses shared libraries (also known as a "dynamically linked" executable). Since we have not installed all of the shared libraries in the initial ramdisk, the program will not run. The second command is particularly important, since we need to transfer the program to the Innovator using tftp. It needs to be placed in the `/tftpboot` directory so that your tftp server can find it.

Once you have executed these two commands, you are ready to transfer the program to the Innovator and to run it using commands similar to the ones shown below (which are to be executed on the Innovator – i.e. in the minicom terminal):

```
# tftp -g -r hello 10.1.0.201
# chmod a+x hello
# ./hello
Hello World
#
```

The first command uses the `tftp` program to get (`-g`) a file named "hello" (`-r hello`) from the tftp server running on the host with an IP address of `10.1.0.201`. (Don't forget to substitute the IP address of your Linux workstation for `10.1.0.201`.) The second command makes the file executable. The third command runs the application. Pretty neat, huh?

Create your own initial ramdisk

Well, now you are probably thinking, "This is great! I am running Linux on my Innovator. The network works. And, I know how to compile, download, and run my own applications. But it sure is a pain to have to type in those IP parameters every time I reboot the Innovator. And, once I am done writing my application, I don't want to have to type in those silly `tftp` and `chmod` commands every time I want to run it. Isn't there an easier way?" The answer is, "Of course there is an easier way!" Actually, there are approximately 546,231,284 easier ways, but we shall only examine one of them. The remaining 546,231,283 easier ways are beyond the scope of this document. The particular easier way we shall examine here is to create your own custom initial ramdisk.

Fortunately, you will not have to create an initial ramdisk from scratch. You can modify the initial ramdisk files that were installed in `~/omaplinux/rootfs`. To make the changes we discussed above, follow the steps listed below:

- The very first time you do this (after unpacking `loti.tar.bz2`), you will have to create device nodes for the Innovator with the following commands:

  ```
  prompt$ su
  password:
  prompt# mknod rootfs/dev/console c 5 1
  prompt# mknod rootfs/dev/ttyS0 c 4 64
  prompt# mknod rootfs/dev/ram0 b 1 0
  prompt# exit
  prompt$
  ```

  The `mknod` command is a UNIX command that creates a "device node" in the file system. Since device nodes are the only way to gain direct, low level, access to devices such as the hard disk, creation of such nodes is restricted to the superuser. This is why the start of the previous sequence of commands is the `su` ("switch user", or "superuser") command. This, incidentally, is also the reason why the `mkrd` script we present below must be executed as root. It indirectly creates device nodes via the `tar` command sequence that is used to copy the contents of the `rootfs` directory to the `initrd.dir` directory.

- Edit `~/omaplinux/rootfs/etc/init.d/rcS` using your favorite text editor. (For what it's worth, our favorite text editor is Emacs.) You will find two lines in that file that have been commented out (with a # at the beginning of the lines) that contain the `ifconfig` and `route` commands we executed previously. Modify those lines appropriately for your environment and remove the # from the beginning of the lines.

- Create `~/omaplinux/rootfs/etc/resolv.conf` containing the nameserver information you manually entered previously.

- Copy your application (i.e. `hello`) to an appropriate location in `~/omaplinux/rootfs`. Logical places are `~/omaplinux/rootfs`, `~/omaplinux/rootfs/bin`, and `~/omaplinux/rootfs/usr/bin`. Note that you should not have to execute the `chmod` command that we executed previously after copying the file from one location on your workstation to another, but it will do no harm to execute it.

- Create a new `initrd` file using commands similar to the ones shown below. Note that most of these commands need to be executed as root.

  ```
  prompt# dd if=/dev/zero of=initrd.img bs=1k count=4096
  prompt# mke2fs -F -m0 initrd.img
  prompt# mount -o loop initrd.img initrd.dir
  prompt# (cd rootfs; tar cf - .) | (cd initrd.dir ; tar xpBf -)
  prompt# umount initrd.dir/
  prompt# gzip -c -9 initrd.img > /tftpboot/initrd
  ```

The `dd` command creates a file named `initrd.img` that contains 4Mbytes of zeros. The `mke2fs` command writes the appropriate headers to that file to create an (empty)“ext2” filesystem. (This is the same command you would execute to create an empty file system on your hard disk.) The `mount` command grafts the empty “ext2” filesystem contained in `initrd.img` onto the filesystem on your workstation under the `initrd.dir` directory. The funny looking `(cd ... tar xpBf -)` command makes a copy of the `rootfs` directory in the `initrd.dir` directory, preserving file permissions, device nodes, etc... (This is a general purpose incantation that can be used any time you want to make an exact duplicate of one directory tree on another). The `umount` command unmounts the filesystem. That is, it removes the “graft” we created with the `mount` command previously. Finally, the `gzip` command compresses the `initrd.img` file (which now contains a valid file system that duplicates the files and subdirectories under the `rootfs` directory) and places the result in the `/tftpboot` directory in a file conveniently named `initrd`. You can use RedBoot to transfer that file to your Innovator using the commands we discussed previously.

Once again, we are going to claim that we know what you are probably thinking. This time, we claim that you are probably thinking, “Gee, do I really have to type those six commands in every time I want to generate a new initial ramdisk? That's going to be a pain, especially that `(cd ... tar xpBf -)` command.” If you are thinking that, then the answer is “no, and you're right, typing that `(cd ... tar xpBf -)` could be a pain, but you actually get used to the pain after a while. But if you don't *really* want to type all of those commands in each time, you don't have to. You can simply run the `mkrd` script (as root) that we conveniently placed in the `~/omaplinux` directory for you.

- In the future, when you make changes to the `rootfs` directory that you want to try out on your Innovator, simply execute the `mkrd` script (as root) as shown in the example below.

```
prompt# ./mkrd
4096+0 records in
4096+0 records out
mke2fs 1.23, 15-Aug-2001 for EXT2 FS 0.5b, 95/08/09
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1024 inodes, 4096 blocks
0 blocks (0.00%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
1024 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 30 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
prompt#
```

- Here is a tip: If you boot the kernel with your new initial ramdisk, but you never see the `Please press Enter to activate this console` prompt, go back and make sure that created the `rootfs/dev/console` device node correcty as described on page 15.

Where to go from here

We hope that this simple introduction has proved informative. Obviously, there are a number of different variations on the theme presented here. The most obvious first step is the selection of a more appropriate root filesystem. The initial ramdisk concept in Linux is just that - an initial file system suitable for loading in necessary modules before initializing the rest of the system. Although there is nothing intrinsicly wrong with using the initial ramdisk for development, you will quickly notice its limitations. For example, it is (with the default configuration) limited to be less than 4 Megabytes in length. Also, the cycle of: compile your application; create a new initial ramdisk (using the `mkrd` script we presented above); reset the board; load the new initial ramdisk via tftp; load the old kernel from the flash; boot the new kernel; gets long and tiresome after a while. Using an NFS mounted root filesystem is an excellent technique for eliminating these tiresome steps, but is beyond the scope of this document. (If you are interested, look at the file `Documentation/nfsroot.txt` in your favorite Linux kernel source tree). Also, for production systems (where you may not have network access to the root file system inconveniently stored on your personal workstation), you should probably investigate other options for the root file system such as ROMFS, CRAMFS, or JFFS2.

Finally, we have deliberately ignored the most exciting part of the OMAP device: the '55x DSP that lives along side of the ARM processor that we have been playing with for the last 10 pages or so. But that is the subject of a different paper...

3. Source Installation

If you have read this far, one of the things you might be thinking is “Gee, it's awfully nice that they provided precompiled binaries for the toolchain, the kernel, and the contents of the initial ramdisk, but I thought that the whole point of Open Source Software was that I would have to compile those pieces from the original source code myself.” As we have just demonstrated, you are not required to compile Open Source Software from scratch. On the other hand, under the terms of the GNU Public License (GPL), which is the license under which the toolchain, kernel, and the contents of the initial ramdisk, we are obligated to make the source code available for the binaries that we distribute.[7] With that bit of trivia in mind, let us see how the files you found in `loti.tar.bz2` were generated.[8]

Compile the Toolchain

The toolchain consists of the compiler, assembler, linker, C runtime libraries, and other miscellaneous utilities that you installed in `/usr/local/xtools` back at the beginning of this document. The compiler we use is gcc-3.2.1. The assembler, linker, and “other miscellaneous utilities” that we use are provided by binutils-2.13.2. The C runtime libraries are provided by uClibc-0.9.17. Erik Andersen, the maintainer of uClibc, has provided a simple mechanism for downloading and compiling all of these independent pieces in one swell foop. We describe that mechanism below.

- Go to http://www.uclibc.org/cgi-bin/cvsweb/toolchain/gcc-3.2.1 and click on the “Download tarball” link near the lower left corner. The instructions that follow assume that you download the tarball to `/dist/omap`. Alternatively, you may choose to download `toolchain-src.tar.bz2`, from the Delphi FTP site: ftp://ftp.delcomsys.com/pub/omap/loti. (Note that all of the source files described in Section 3 are available on the Delphi FTP site in addition to the official source sites described here)
- Create a working directory for building the toolchain. The instructions that follow assume that the toolchain directory is `~/xtools`.
- Unpack the `gcc-3.2.1.tar.gz` tarball:

```
prompt$ mkdir ~/xtools
prompt$ cd ~/xtools
prompt$ tar xzf /dist/omap/gcc-3.2.1.tar.gz
prompt$ cd gcc-3.2.1
```

7 We are not obligated to write a document describing how to use that source code, that's just a bonus feature you get by reading this document.

8 This section omits the instructions for generating RedBoot. See our separate document about running eCos on the Innovator for a description of how to compile RedBoot.

- Modify the top level `Makefile` to suit our installation. (Note that, if you fetch `gcc-3.2.1.tar.gz` from `www.uclibc.org`, some of these modifications may have already been made for you. If you fetch `toolchain-src-tar.bz2` from `ftp.delcomsys.com`, all of these modifications will have been made).[9]
    - Set `TARGET_PATH` to `/usr/local/xtools/arm-uclibc-3.2.1`
    - Set `ARCH` to `arm`
    - Set `USE_UCLIBC_SNAPSHOT` to `false`
    - Change `uClibc-0.9.16` to `uClibc-0.9.17` in the macro definitions for `UCLIBC_SOURCE` and `UCLIBC_DIR`
- Now build the toolchain. This will take a while. (And, will take even longer if you fetched `gcc-3.2.1.tar.gz` from `www.uclibc.org` because, unlike the version at `ftp.delcomsys.com`, Erik's version does not include the distribution files for binutils, gcc, and uClibc. It downloads them automatically as part of the build process. This is why his version downloaded significantly faster than the version at `ftp.delcomsys.com`.)

  ```
  prompt$ make > makelog 2>&1 &
  prompt$ tail -f makelog
  ```

  (lots of output deleted - press ^C to exit the `tail` program).

  The first command listed above executes the `make` program, but directs its output (`>`), including its error output (`2>&1`), to a file named `makelog`. (Note that this syntax for directing stdout and stderr is specific to the Bourne shell (i.e. bash)). The second command simply monitors the contents of the `makelog` file so you can see what is happening and notice when it completes. You will notice when it completes because it will display “Finally finished!”[10]
- Don't forget to add `/usr/local/xtools/arm-uclibc-3.2.1/bin` to your path. Most people add this in their `.bashrc` or `.cshrc` file so that it is always available.

9 Conversely, it is possible that, if you fetch the latest, greatest version from Erik's website, you will get other modifications that do not match what is presented here. But, if you want the official source for the toolchain, Erik's site is the place to go.

10 Alternatively, you could execute the command: `make 2>&1 | tee makelog`, but this form allows you to halt the `tail` command and disconnect from the machine to which you are remotely logged in (using SSH right?) and go home at the end of the day.

Compile the kernel

Now that you have a toolchain, you are ready to compile the kernel. In order to start from scratch, you must obtain three files: the official 2.4.19 Linux kernel source tree; the official ARM patch for that tree; and the official OMAP patch for the ARM patch source tree. The kernel source is available at http://www.kernel.org. Make sure you download `linux-2.4.19.tar.bz2` if you want to follow the directions we give precisely. The ARM patch is available at http://www.arm.linux.org.uk/developer/v2.4/. Make sure you download `patch-2.4.19-rmk4.bz2` if you want to follow the directions we give precisely. Finally, the OMAP patch is available at ftp://source.mvista.com/pub/omap/patch-2.4.19-rmk4-ggd2.bz2. The instructions that follow assume that you have downloaded these files to `/dist/omap` on your Linux workstation and that you use `bash` as your shell.

- Unpack and patch the kernel. The instructions presented below assume that you unpack the kernel in your `~/omaplinux/kernel` directory.

  ```
  prompt$ mkdir ~/omaplinux/kernel
  prompt$ cd ~/omaplinux/kernel
  prompt$ tar xjf /dist/omap/linux-2.4.19.tar.bz2
  prompt$ cd linux-2.4.19
  prompt$ bzcat /dist/omap/patch-2.4.19-rmk4.bz2 | patch -p1
  prompt$ echo $?
  ```

  Note that the `echo $?` command checks to see if the patch was successful. It should display 0 as its result. If it does not, then make sure that you typed in the commands precisely as they were shown.

  ```
  prompt$ bzcat /dist/omap/patch-2.4.19-rmk4-ggd2.bz2 | patch -p1
  prompt$ echo $?
  ```

  These commands create a working directory (named `~/omaplinux/kernel`); unpack the baseline source tree into it; and apply the patchfiles to it. In the interest of brevity, we have omitted the output from the two patch commands, so don't be surprised when you see mounds of output from those two commands. Note that all of the steps that follow assume that the current working directory is `~/omaplinux/kernel/linux-2.4.19`.

- There are still a few more patches to be made, this time by hand. These steps will not be required once these patches are fed back into the OMAP and ARM source trees.
  - Edit `linux-2.4.19/arch/arm/mach-omap/innovator.c` and add a line that reads `BOOT_PARAMS(0x10000100)` after line 358. For reference, line 358 should read: `BOOT_MEM(0x10000000, 0xe0000000)`.
  - Edit `linux-2.4.19/drivers/net/Config.in` and change `dep_tristate` to `tristate` on lines 29 and 112.

- We are *almost* ready to start building your custom kernel. The next step is to make two changes to the top level makefile (in `linux-2.4.19/Makefile`). (I know, you thought you were done making patches. Sorry.) Find the line that sets the `ARCH` macro and change it to read:

  ```
  ARCH := arm
  ```

  Then find the line that sets the `CROSS_COMPILE` macro and change it to read:

  ```
  CROSS_COMPILE     = arm-uclibc-
  ```

- Now it is time to install the baseline configuration file for the Innovator. The next two commands copy the baseline configuration file to `.config` and process it to make sure that it is compatible with the current kernel release.

  ```
  prompt$ make innovator_config
  prompt$ make oldconfig
  ```

- Run the graphical Linux Kernel Configuration tool to modify a few parameters to match the setup presented in this paper. When you execute the next command, you will be presented with a new X window showing the *Linux Kernel Configuration* options. (Note that this assumes that you are running the X windowing system on your Linux workstation. If you are not, you can still configure the kernel, but you will have to read a different paper.)

  ```
  prompt$ make xconfig
  ```

  - Click on the button labeled *General Setup*. (It should be the fourth button down on the left.) This will result in the display of a new X window labeled *General Setup*.
  - Use the scroll bar (in the new window) to scroll through the list of options to the end. Search for the option labeled *Default kernel command string*. You will see (part of) the default value in the option box to the left of the label. By default, it contains "`mem=32M console=ttyS0,115200n8 noinitrd root=/dev/nfs ip=bootp`" Change it so that it reads " `console=ttyS0,115200n8`" (i.e. remove everything from the default value except for the `console=...` part). This configures the kernel so that its startup console is the serial port running at 115200 baud, no parity, and 8 bits per character.
  - Click on the button labeled *Main Menu*.
  - Click on the button labeled *File systems*. (It should be the last button in the middle column). This will result in the display of a new X window labeled, by a bizarre set of circumstances, *File systems*.
  - Use the scroll bar to scroll through the list of options, almost to the end, to find the option labeled *Second extended fs support* (also known as "ext2"). On the left hand side of the window, you should see that the "m" option is selected (indicating that ext2 support is provided via a loadable module). Click on the "y" option (indicating that yes, you want ext2 support to be built into the kernel). This enables support for the second extended file system to be built into the kernel directly instead of a as a separately loaded module. The initial ramdisk that we create later

is created as an ext2 filesystem, so support for this type of filesystem must be built into the kernel.

- Click on the button labeled *Main Menu*.
- Click on the button labeled *Save and Exit*.
- Click on the button labeled *OK*.

Congratulations! You have just configured your first Linux kernel for the Innovator. Once you verify that the kernel works (which we shall do shortly), feel free to return to the configuration tool and browse the options. Some of them will, undoubtedly, provoke interesting thoughts and ideas. Go ahead, play!

- Now that you have configured your kernel, it is time to build it. First you must update the dependencies (which is not really required for a clean install, but it is a great habit to get into), and then you can build the kernel image itself. We use the following two commands (using our trick of logging the output to a file, and watching that file, instead of wishing that we could have seen what the first three lines of output were long after they have scrolled off of the screen):

```
prompt$ make dep Image > makelog 2>&1 &
prompt$ tail -f makelog
```

- Once the build completes, you are ready to copy it to `/tftpboot` and try it out using your existing (presumably known to be working) initial ramdisk:

```
prompt$ cp arch/arm/boot/Image /tftpboot
```

Load the new kernel on your Innovator using RedBoot as we described in Section 2.

Create the initial ramdisk

In Section 2 of this document, we described the process of creating an initial ramdisk image for the Innovator from an existing directory structure on your workstation. We even discussed how to modify or add files to that directory structure, but we carefully did not discuss how that directory structure was created in the first place. Now we are ready to show you how it was created. Most of the directory structure was created as a result of configuring, building, and installing a single application called BusyBox. BusyBox is the self proclaimed "Swiss Army Knife of Embedded Linux". It is a single application that "... combines tiny versions of many common UNIX utilities into a single small executable. It provides minimalist replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc. The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts. BusyBox provides a fairly complete POSIX environment for any small or embedded system. " In particular, BusyBox provides an implementation of the typical UNIX utilities such as `ls` and `more` as well as the `ifconfig` and `route` commands we typed in (and later added to the `/etc/init.d/rcS` file) in the previous section.

- The first step is to create the empty root file system and a few useful directories. However, before performing this step, you should remove/rename the old `rootfs` directory, as shown in the commands below:

  ```
  prompt$ cd ~/omaplinux
  prompt$ mv rootfs rootfs.old
  prompt$ mkdir -p rootfs/{etc/init.d,dev,proc} initrd.dir
  ```

  The `mkdir` command creates a total of six directories: `rootfs`, `rootfs/etc`, `rootfs/etc/init.d`, `rootfs/dev`, `rootfs/proc`, and `initrd.dir`. (The last directory listed doesn't contribute to the filesystem going onto the device, but we will need it, and, since we are busy creating directories anyway, we might as well create it here). We use a nifty feature of the bash shell to save us some typing by grouping all of the subdirectories of `rootfs` together inside the curly braces.[11] If you use csh (or one of its derivatives) as your shell, your mileage may vary. We also use a nifty feature of the `mkdir` command wherein it will create an entire path of directories when given the `-p` option. (Otherwise, we would have had to make the `rootfs` directory first, then the `rootfs/etc` directory, then the `rootfs/etc/init.d` directory, etc...)

- Create the `rootfs/etc/fstab` file containing the following:

  ```
  # <file system>      <mount point>   <type>  <options> <dump>  <pass>
  /dev/ram0            /               ext2     defaults  1       1
  proc                 /proc           proc     defaults  0       0
  ```

  This file is used by the `mount` command to determine which devices (or pseudo-devices) should be mounted on which directories. The first line is a comment.

11 This is shorthand for `rootfs/etc/init.d rootfs/dev rootfs/proc`.

The second and third lines specify the parameters for mounting the root and proc filesystems.

- Create the `rootfs/etc/init.d/rcS` file containing something similar to the following. Note that you must change the network parameters in the last two lines (and remove the comment characters) to match your local network settings as shown previously in Section 2.

  ```
  #!/bin/sh
  /bin/mount -n -o remount,rw /

  /bin/mount -a
  echo Starting Network

  /sbin/ifconfig lo 127.0.0.1 netmask 255.0.0.0 broadcast 127.255.255.255
  /sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo
  #/sbin/ifconfig eth0 10.1.3.2 netmask 255.255.0.0 broadcast 10.1.255.255
  #/sbin/route add default gw 10.1.0.100 eth0
  ```

  This file is executed by the `/sbin/init` program when the kernel boots. (Actually, it is executed by the `/linuxrc` program, but we won't go into that here.) It contains commands to initialize whatever you need to initialize at boot time. None of the commands are, strictly speaking, *required*, but, as we saw in Section 2, if you don't want to set up the network by hand every time you boot Linux on your Innovator, you should add the commands to this file. The same is true if, for example, you wanted to use `tftp` to fetch a program from your workstation each time you boot, although, at that point, an NFS mounted root filesystem, with shared library support, would probably be more appropriate. (That last bit is beyond the scope of this document. Don't you get tired of reading that?)

- Since the `rootfs/etc/init.d/rcS` file is *executed* by `init`, it must be made executable with the `chmod` command as shown in the command below:

  ```
  prompt$ chmod a+x rootfs/etc/init.d/rcS
  ```

  If you forget this step, you will see a funny warning message the next time you boot with your new initial ramdisk, but none of the commands in your `/etc/init.d/rcS` will have been executed. (Trust us on this one – or try it yourself and see...)

- Create the device nodes with the following commands:

  ```
  prompt$ su
  password:
  prompt# mknod rootfs/dev/console c 5 1
  prompt# mknod rootfs/dev/ttyS0 c 4 64
  prompt# mknod rootfs/dev/ram0 b 1 0
  prompt# exit
  prompt$
  ```

- Back at the beginning of this section, we discussed an application called BusyBox, and then proceeded to ignore it for lo these many pages. Now is the time that we finally get to do something with it. BusyBox is available at http://www.busybox.net (as well as at `ftp.delcomsys.com`). The directions that follow assume that you have downloaded `busybox-0.60.5.tar.bz2` to the `/dist/omap` directory.

- Unpack the BusyBox application:

```
prompt$ mkdir ~/omaplinux/userapps
prompt$ cd ~/omaplinux/userapps
prompt$ tar xjf /dist/omap/busybox-0.60.5.tar.bz2
```

- Edit the `busybox-0.60.5/Config.h` file to enable the following options. In each case, the option is commented out (with a C++ // style comment line) and must be enabled by removing the comment characters at the beginning of the line:

```
BB_IFCONFIG
BB_PING
BB_ROUTE
BB_TFTP
BB_FEATURE_IFCONFIG_STATUS
```

- Build the BusyBox applications and install them in your root filesystem:

```
prompt$ cd busybox-0.60.5
prompt$ make CROSS=arm-uclibc- DOSTATIC=true PREFIX=../../rootfs all \
install > makelog 2>&1 &
```

  (Note that the last command is continued on a second line. You may type it all on one line, omitting the trailing backslash, or split it across two lines as shown, provided that you press the return key immediately following the trailing backslash.)

```
prompt$ tail -f makelog
```

Congratulations! You have just configured your first BusyBox application. You are now ready to regenerate the initial ramdisk from the contents of the `rootfs` as described in Section 2.

4. Conclusion and Resources

This document has walked you through the steps of installing Linux on your Innovator, from precompiled binaries and from raw source code. As you can see, there is a lot more to running Linux than simply downloading the source code and compiling it. In addition to the kernel, you need a toolchain capable of compiling the kernel (and, ideally, the applications that will run on top of the kernel), a root file system of some type (we present an initial ramdisk as the root filesystem), and applications that live on the root file system. Once you have all of those tools in place, you will be ready to develop your Linux based application for the OMAP.

Hopefully this has given you enough of an introduction that you can pursue your own Linux system integration on your own, or enough knowledge to decide whether it makes more sense to outsource the Linux integration and enable you to focus on your application.

For your convenience, all of the online resources referenced in this paper (as well as a few that are not) are listed below.

- **Linux Kernel Resources**
  - The Linux Kernel Archives

    http://www.kernel.org
  - The Linux Documentation Project

    http://www.tldp.org
  - The ARM Linux Project

    http://www.arm.linux.org.uk

- **Toolchain Resources**
  - uClibc

    http://www.uclibc.org
  - The glibc based toolchain recommended by the ARM Linux developers

    ftp://ftp.arm.linux.org.uk/pub/linux/arm/toolchain/cross-2.95.3.tar.bz2
  - The Emdebian toolchain

    http://www.emdebian.org/downloads.htm

- **Other Resources**
  - Texas Instruments OMAP homepage

    http://focus.ti.com/omap/docs/omaphomepage.tsp
  - eCos

    http://sources.redhat.com/ecos
  - Linux From Scratch

    http://www.linuxfromscratch.org
  - The "From Power Up To Bash Prompt" HOWTO

    http://www.netspace.net.au/~gok/power2bash
  - uClinux
    http://www.uclinux.org
  - cross-GCC FAQ
    http://www.objsw.com/CrossGCC/
  - Win4Lin
    http://www.netraverse.com/

- **OMAP Technology Centers supporting Linux**
  - Delphi

    http://www.delcomsys.com