

Building and Installing Software for the Mini2440

Jim Donelson

Building and Installing Software for the Mini2440

Jim Donelson

Acknowledgements

Thanks to Bill of Bill's blog and Bill's Mini2440 Forum for most of the information here. Bill claims to be an "installer", but he also seems to have dogged determination to figure this stuff out. So he his not a documentor, and while I may not be the best at this, but this is my attempt to document and compile his experiences and notes. One tough thing about having this information in the blog and forum is that if you don't read the entire thread (mindless chatter and all), you may miss some important corrections/insights, so I have attempted to remedy that.

Note that I have personally tried everything herein, and need your feedback to improve it.

Table of Contents

1. Overview	4
1.1. The Big Picture	4
1.1.1. Toolchains and Cross Compiling	4
1.1.2. UBoot	5
1.1.3. Kernel	5
1.1.4. Root File System	5
1.1.4.1. A Requirement	5
1.1.4.2. Busybox	5
1.1.5. Qt	5
1.2. Openembedded and Bitbake	5
1.2.1. Bitbake	5
2. Install the Prerequisites	7
2.1. Introduction	7
2.2. Packages	7
2.3. Install TFTP Server	8
2.4. Apt-get List	8
2.5. Setting up picocom	9
2.6. Using the USB Cable in Linux	9
3. Das U-Boot - Universal Bootloader	10
3.1. Why do this?	10
3.2. Toolchain	10
3.3. Building	10
3.4. Flashing UBOOT to NAND	11
3.5. Other UBOOT Stuff	11
4. Building the Kernel	12
4.1. Which Kernel?	12
4.1.1. Option 1 - The OEM Kernel	12
4.1.2. Option 2 - The "other" Kernel	12
4.2. Setting up Your Toolchain	12
4.3. Building Option 1 OEM Kernel	13
4.4. Obtaining Kernel Sources for Option 2	14
4.5. Compiling Your Kernel	15
4.6. Installing Kernel Modules To Your File system	15
4.7. Customizing Your Kernel Configuration	15
4.8. Cleaning your source tree	15
4.9. Final Thoughts	15
5. Building and Installing Openembedded	17
5.1. Setting up the Directory Structure	17
5.2. Install Bitbake	17
5.3. Git Openembedded	17
5.4. Prepare the Config File	17
5.5. Set BBPATH and Kill dash	18
5.6. Building Openembedded Recipes	18
6. Compiling Qt and Tslib	21
6.1. Set up the toolchain	21
6.2. Build TSLIB	21
A. Formatting the SD card	22
A.1. Mount the SD card	22
A.2. Partition the card	22
A.3. Format the card	22

List of Examples

2.1. Install TFTP	8
2.2. Apt Get List	9
2.3. Installing and Running picocom	9

Chapter 1. Overview

1.1. The Big Picture

It's easy to get so caught up in the detail and forget why we need to drain the swamp, so this chapter is to try and put a sense of order and purpose the the tasks.

1.1.1. Toolchains and Cross Compiling

You will probably be working on an x86 machine. When you type 'gcc' you are using a compiler that generates machine level code for an x86 processor. This is referred to as a 'native' compiler, as it is generating code for the machine it is running on. We need to generate code for the ARM architecture, so we use a cross compiler, that runs on an x86 machine but generates code for the ARM machine. This is referred to as the target machine.

It goes beyond just the compiler, we also need a binutils (linker and assembler) and a glibc ("C" runtime libraries) that is compatible with the target. There is no one toolchain that we will use, as unfortunately gcc evolves and certain items need certain arm toolchains to properly build. We will be installing several.

There is a little trick to cross compiling. The toolchains create programs that have a prefix, for example for gcc:

```
arm-none-linux-gnueabi-gcc
```

So build system knows that if you give it the prefix, arm-none-linux-gnueabi- (in this case), that appending this to compiler name and all the other utilities will invoke the proper program. (This is a CodeSourcery toolchain from 2008)

There is another variable ARCH that usually need to be defined as well.

So, something like this is done to set up a build, in this case the kernel:

```
You define:
CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- ARCH=arm

Some where in the Makefile:
AS      = $(CROSS_COMPILE)as
LD      = $(CROSS_COMPILE)ld
CC      = $(CROSS_COMPILE)gcc
CPP     = $(CC) -E
AR      = $(CROSS_COMPILE)ar
NM      = $(CROSS_COMPILE)nm
STRIP   = $(CROSS_COMPILE)strip
OBJCOPY = $(CROSS_COMPILE)objcopy
OBJDUMP = $(CROSS_COMPILE)objdump
Then, to compile a file, the Makefile would:
$(CC) myfile.c
To run the assembler:
$(AS) code.S
```

There are several cross compilers we may be using. It might seem strange that you would need more than one toolchain, but it is best to use the one originally used to build with.

Installing a tool chain:

The tool chain from FirendlyARM

```
$ sudo tar xfvz arm-linux-gcc-4.3.2.tgz -C /
```

1.1.2. UBoot

The boot loader is a piece of code that set up the board, finds the kernel, loads it and starts it. UBOOT is a very versatile option for this purpose, and provides many ways to boot up the board. It can boot from flash, SD card and the network.

1.1.3. Kernel

The kernel is the operating system. No doubt you are well aware of what the kernel is ????. Currently I am using the kernel provided by FriendlyARM dated 2010-06-09, which can be downloaded from the FriendlyARM website. This provides for much better board support than other versions of mini2440 kernels. To use this one we will need to adjust the configuration some, and this will be a great change to learn how to do this. There are two tool I use for this 'menuconfig' which is a terminal style configuration utility and 'xconfig' which a GUI tool. While the GUI tool is by far the easiest, if you ever need to do a configuration over telnet, it is a good idea to be familiar with 'menuconfig'

1.1.4. Root File System

1.1.4.1. A Requirement

A running Linux kernel must a root file system. This is actually to most complicated part of the job and for this I am using the Openembedded build of Angstrom. There are a few really cool things, one of them is a package manager, along the lines of apt-get, that will allow you to install packages right on the mini2440.

1.1.4.2. Busybox

Busy box is a package that provides all the basic things required for a root file system in a very compact form. At first we will use what comes with Angstrom, but we will re-build.

1.1.5. Qt

We will re-build QT and tslib so that we have an uptodate version. This is the last step.

1.2. Openembedded and Bitbake

Currently this is the latest tool used for build a complete distro. It will build the toolchain, Kernel,root file system and much more. In fact perhaps too much more.

From the web site:

Openembedded was invented and founded by the creators of the OpenZaurus project. At this time the project had pushed buildroot to its limits. It supported the creation of ipk packages, feeds and images and had support for more than one machine. But it was impossible to use different patches, files for different architectures, machines or distributions. To overcome this shortcoming OpenEmbedded was created.

After a few months other projects started using OpenEmbedded and contributing back. On 7 December 2004 Chris Larson split the project into two parts: BitBake, a generic task executor and OpenEmbedded, the metadata for BitBake.

1.2.1. Bitbake

From the web site:

BitBake is, at its simplest, a tool for executing tasks and managing metadata. As such, its similarities to GNU make and other build tools are readily apparent. It was inspired by Portage, the package management system used by the Gentoo

Linux distribution. BitBake is the basis of the OpenEmbedded project, which is being used to build and maintain a number of embedded Linux distributions, including OpenZaurus and Familiar.

Chapter 2. Install the Prerequisites

2.1. Introduction

This helps you the set up things you will need on you system for embedded development.

2.2. Packages

This was done on a fresh install of Ubuntu 10.10. It's possible there could be some missing packages, if you find one please update us.

- git =>1.6.3.3
- bitbake =>1.8.18
- u-boot mkimage
- bison
- flex
- e2fsprogs - Ext2/3/4 Filesystem Utilities
- m4
- curl - command line tool for transferring data with URL syntax
- ncurses libraries and development headers
- zlib development libraries and headers
- cvs
- subversion
- unzip
- bzip2 and its development libraries and headers
- gcc 3.x On Ubuntu try installing (in this order) :
 1. gcc-3.4-base_3.4.6-8ubuntu2_i386.deb
 2. cpp-3.4_3.4.6-8ubuntu2_i386.deb,
 3. gcc-3.4_3.4.6-8ubuntu2_i386.deb
- /proc/sys/vm/mmap_min_addr needs to be 0. (or oe will complain)
- qt3-dev-tools runtime for xconfig

Open Embedded will complain if mmpa_min_addr is not set to zero,

From the Debian Wiki:

mmap_min_addr is a kernel tunable that specifies the minimum virtual address that a process is allowed to mmap. Allowing processes to map low values increases the security implications of a class of defects known as "kernel NULL pointer dereference" defects. If a malicious local user finds a way to trigger one of these NULL pointer defects, they can exploit it to cause system hangs, crashes, or otherwise make parts of the system unusable. If this user is also able to map low portions of virtual memory, they can often further exploit this issue to gain increased privileges.

You can set `mmap_min_addr` like this:

```
/oe$ sudo -i
root@ubuntu:~/oe# echo 0 > /proc/sys/vm/mmap_min_addr ; exit
```

Also edit `/etc/sysctl.conf` to permanently set it:

```
$ sudo vi /etc/sysctl.conf
(Change or add the line: vm.mmap_min_addr=0)
```

2.3. Install TFTP Server

It is quite handy to have TFTP, because UBOOT can fetch file using it.

Example 2.1. Install TFTP

```
sudo apt-get install xinetd tftpd tftp

$ sudo mkdir /tftpboot
$ sudo chmod -R 777 /tftpboot
$ sudo chown -R nobody /tftpboot
$ sudo /etc/init.d/xinetd restart

Is tftp running?
sudo netstat -nap | grep :69
udp        0      0 0.0.0.0:69          0.0.0.0:*
1760/in.tftpd

Test it

$ touch /tftpboot/hda.txt
$ chmod 777 /tftpboot/hda.txt
$ ls -l /tftpboot/
total 0
-rwxrwxrwx 1 davids davids 0 2006-03-27 23:04 hda.txt
(create a file called hda.txt)

$ tftp <your ip address>
tftp> put hda.txt
Sent 722 bytes in 0.0 seconds
tftp> quit
$ ls -l /tftpboot/
total 4
```

2.4. Apt-get List

This should set up most of the packages you need:

Example 2.2. Apt Get List

```
sudo apt-get install ccache sed wget cvs subversion git-core monotone \
  coreutils unzip texi2html texinfo libsdl1.2-dev docbook-utils \
  gawk python-pysqlite2 diffstat help2man bison flex m4 python-psyc \
  curl subversion e2fslibs-dev uboot-mkimage g++ qt3-dev-tools
```

2.5. Setting up picocom

Chances are you are using a USB to serial adaptor. Most common USB to serial adaptors have built in drivers, and after you plug it in, run DMSG to find it.

```
$ dmesg
```

```
[954264.608244] usb 2-1: new full speed USB device using uhci_hcd and address 7
```

```
[954264.779413] pl2303 2-1:1.0: pl2303 converter detected
```

```
[954264.806461] usb 2-1: pl2303 converter now attached to ttyUSB0 j
```

We see that the port came up as `/dev/ttyUSB0`.

Example 2.3. Installing and Running picocom

```
ls /dev | grep ttyU
ttyUSB0
sudo apt-get install picocom
picocom -b 115200 /dev/ttyUSB0
man picocom          (for more help)
```

To exit picocom use `ctrl-A ctrl-X`

2.6. Using the USB Cable in Linux

http://mini2440.googlecode.com/files/s3c2410_boot_usb-20060807.tar.bz2

```
tar xvf s3c2410_boot_usb-20060807.tar.bz2
```

```
sudo apt-get install libusb-dev
```

Change `CHUNK_SIZE` from 100 to 32000 in `boot_usb.c`

```
make
```

```
load ram 0x32000000 242332 u
```

```
sudo ./s3c2410_boot_usb u-boot-128M.bin
```

Chapter 3. Das U-Boot - Universal Bootloader

This assumes you already have the Kernel and root file system ready to go on an SD card. Don't do this until you do, or you will not have anything to boot to.

3.1. Why do this?

You really don't need to build UBOOT. The UBOOT on the FriendlyARM site work just fine, and so far I see no advantage to building it.. Download U-Boot 2010-07-01 which is a zip file. However, UBOOT supports booting from other file systems and is the best way to go.

Unzip the file, and use 128M or 256M - for a 1G mini2440 I used the 256M version and it worked fine. If you go this route, skip to the Flash NAND section.

(I don't have a 64M NAND board, so I don't know if it will work on a 64M board)

3.2. Toolchain

Ok, so you've at least decided to see what's involved. This assumes you have installed all the prerequisite packages. (You should be able to click the link and get it).

<http://www.codesourcery.com/sgpp/lite/arm/portal/package3696/public/arm-none-linux-gnueabi/arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2>

<http://www.codesourcery.com/sgpp/lite/arm/portal/package3696/public/arm-none-linux-gnueabi/arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2>

To install it:

```
$ wget http://www.codesourcery.com/sgpp/lite/arm/portal/package3696/public/ \
  arm-none-linux-gnueabi/arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
$ sudo tar xjvfo \
  arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 \
  -C /usr/local
$ export PATH=$PATH:/usr/local/arm-2008q3/bin
```

You may also wish to add this to your .bashrc file.

3.3. Building

Create a directory in your home folder and git the source:

```
mkdir uboot ; cd uboot
git clone git://repo.or.cz/u-boot-openmoko/mini2440.git
export CROSS_COMPILE=arm-none-linux-gnueabi-
```

I have omitted a section in which the USB was disabled, because now the source comes like that, plus the way I flash it is not an issue.

```
make mini2440_config  
make
```

3.4. Flashing UBOOT to NAND

I use the method of just flashing right to NAND using the BIOS in NOR and DNW. The load to RAM has never worked for me, and really why bother?

You will need to connect to the board with a terminal program and DNW running.

1. Switch the system to NOR and boot with supervivi.
2. Hit 'v' in supervivi
3. Uploaded the u-boot.bin file using DNW.
4. Switch the system to NAND and boot up with u-boot.
5. Run

```
MINI2440# nand scrub  
MINI2440# nand createbbt  
(note: On a 1G Flash system this can take up to 20 minutes - let it finish)  
MINI2440#
```

6. Switch back to NOR, because you just erase flash and need to burn NAND again
7. Hit 'v' again and upload uboot again, as you erased it with the scrub.
8. switch back to nand and it should boot up with minimal errors using u-boot.
9. Run:

```
MINI2440# dynpart  
MINI2440# saveenv  
MINI2440# dynenv set u-boot_env  
MINI2440# saveenv
```

Important

DNW Can be flakey at times. I usually have plug and un-unplug the board a few times. Also uploads sometime just die on me, so if it does not seem to work try again.

3.5. Other UBOOT Stuff

You should set the MAC address. I like to use Steve Jobs old prefix from his NEXT company. Fill in the 00:00:02 with what ever you wish,

```
set ethaddr 00:00:ff:00:00:02
```

Chapter 4. Building the Kernel

4.1. Which Kernel?

I believe the best Kernel to use is the one from the FriendlyARM site. It works fine with the OpenEmbedded root file system, and fully supports all the hardware on the board.

4.1.1. Option 1 - The OEM Kernel

Option 1 is using the Kernel from the FriendlyARM site. This is the file `linux-2.6.32.2-mini2440_20100609.tgz`.

http://www.friendlyarm.net/dl.php?file=linux-2.6.32.2-mini2440_20100609.tgz

4.1.2. Option 2 - The "other" Kernel

This is a Kernel that was put together before the OEM Kernel was out, and lacks most of what you need to use the board.

4.2. Setting up Your Toolchain

Both options build with this toolchain.

This is the tool chain we will use (*it's about 80MB*) for both kernels:

```
wget http://www.codesourcery.com/sgpp/lite/arm/portal/package3696/public/\
arm-none-linux-gnueabi/arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
```

To install it:

```
sudo tar xjvfo arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2\
-C /usr/local
export PATH=$PATH:/usr/local/arm-2008q3/bin (add this to ~/.bashrc as well)
```

When you type:

```
arm-none-linux-gnueabi-gcc -v
```

You should get about half a screen of chatter. It should end in: `gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)`

This will create a directory called `arm-2008q3` in your `/usr/local` directory and add the tool chain to your users path.

Since you've already added it to your `bashrc`, from this point on, you shouldn't need to add it to your path anymore. If for some reason you're not using `bash`, or you this doesn't work for you, you can manually run the above 'export' command each time you want to compile and it will have the same affect.

Next, you should set your `CROSS_COMPILE` environment variable to match the prefix on the toolchain. If you've looked in the toolchain's bin directory, you've probably noticed a bunch of files beginning with `arm-none-linux-gnueabi-` .. Well, that's what we set the variable to.

```
export CROSS_COMPILE=arm-none-linux-gnueabi-
```

Now anything that you compile that has a properly created Makefile and is capable of being cross compiled will use that toolchain. You'll need to set that variable each time you want to compile. But I don't recommend adding it to your bashrc file since you won't always want to cross compile everything.

You can now start using OpenEmbedded. *(really? Openembedded has it own toolchain that it uses)*

4.3. Building Option 1 OEM Kernel

We will need to config this Kernel to add some things we need. You can use menuconfig or xconfig, or any other one you wish, but I recommend xconfig. If you installed all the recommended packages both fo there should work.

So the source is linux-2.6.32.2-mini2440_20100609.tgz from the FriendlyARM site.

```
$ mkdir mini2440-OEM-kernel
$ cp (path)/linux-2.6.32.2-mini2440_20100609.tgz mini2440-OEM-kernel
$ cd mini2440-OEM-kernel
$ tar xfvz linux-2.6.32.2-mini2440_20100609.tgz
$ cd linux-2.6.32.2
$ ls
```

We need to copy one of these to "dot-config" (.config). I have the 3.5" LCD, so I used config_mini2440_t35

- config_mini2440_a70
- config_mini2440_l80
- config_mini2440_n35
- config_mini2440_t35
- config_mini2440_vga1024x768

```
$ cp config_mini2440_t35 .config
```

Create a file call cross.sh, and the chmod +x it:

```
CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- \
ARCH=arm INSTALL_MOD_PATH=/home/YOURUSERNAME/mini2440-OEM-kernel make $1

(make the above all one line)
```

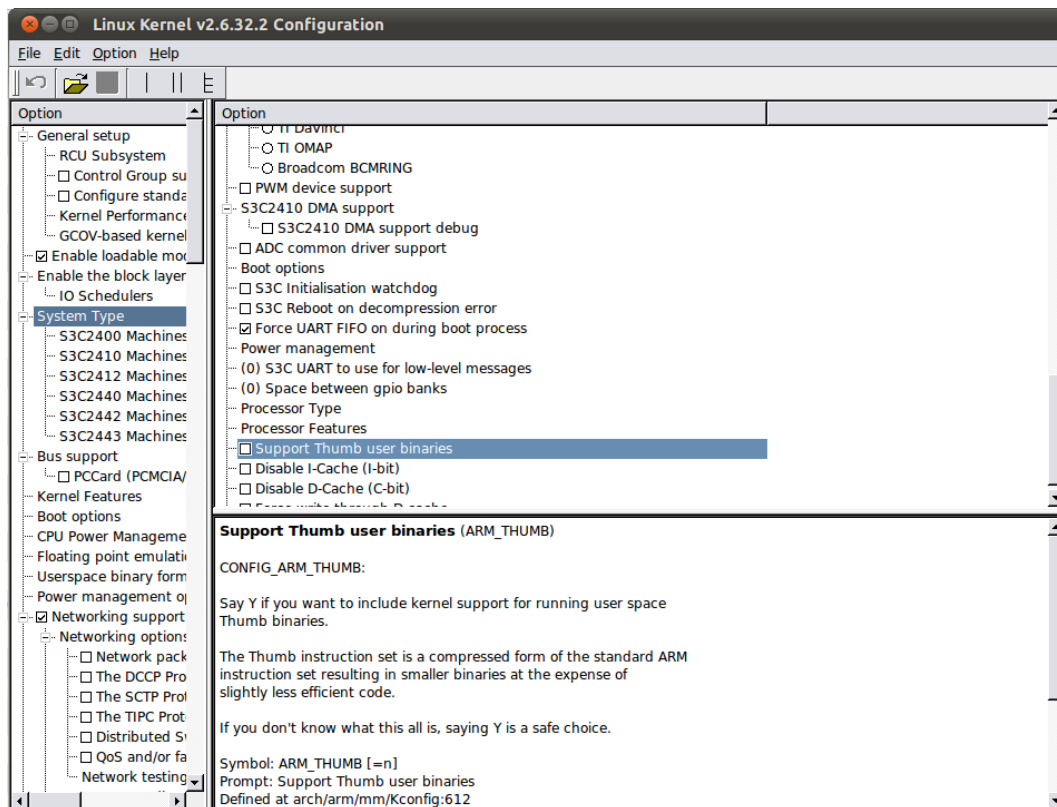
Then you can use this for the following.

```
$ ./cross.sh xconfig
(this is the same as)
CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- \
ARCH=arm INSTALL_MOD_PATH=/home/jdonelson/mini2440-org-kernel make xconfig
```

There are serverl type of configuration options. SOme are a string, some are yes/no and some are tri-state. For a tri-state a dot means loadable module, and a check means build in.

The screen shot below shows how xconfig should look, and is at the first option we need to add, "Support thumb user binaries". Also, under "General Setup" change Local Version to your initials or a revision number. I used 'jcd.R1' or 'batman'.

Click it so it is checked.



We also need to turn on EXT2, EXT3 and JFFS2 file systems. These are Kernel drivers that can be set as built in or loadable modules. We want these built in, so we do NOT want the dot, we want a check mark for that option.

When you have it all configured:

```
$ ./cross.sh uImage
$ ./cross.sh modules
```

We need uImage for UBOOT. It will be in arch/arm/boot/

Next you need to build the modules directory:

modules_install

```
$ ./cross.sh modules_install
```

This will create a directory called lib in mini2440-OEM-kernel that needs to be copied to the root file system.

4.4. Obtaining Kernel Sources for Option 2

This is to build

Create a directory for your source tree and change to it and use git to clone the repository.

```
$ mkdir mini2440-kernel ; cd mini2440-kernel
$ git clone git://repo.or.cz/linux-2.6/mini2440.git (this could take .5hr-1hr)
$ cd mini2440
```


4.5. Compiling Your Kernel

The first step in compiling is to create the default configuration file. (about 12 mins on my box)

```
$ CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- ARCH=arm make mini2440_defconfig
```

When this is complete. Your kernel is compiled.

Now we need to create a uImage. Uboot requires this type of image, that's why it's call uimage.

```
$ CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- ARCH=arm make uImage
```

Your uImage for booting your device is in arch/arm/boot and you can copy that to your SD card, or NFS directory or NAND.

4.6. Installing Kernel Modules To Your File system

It's very simply to install modules into your file system for the mini. Let's say your SD card has your file system for the mini2440, and it's mounted in /mnt

```
$ CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- ARCH=arm INSTALL_MOD_PATH=/mnt make modules
```

4.7. Customizing Your Kernel Configuration

```
$ CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- ARCH=arm make menuconfig
```

4.8. Cleaning your source tree

```
$ CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- ARCH=arm make clean
```

This will simply clean the tree of binaries and compiled object files etc but leaves the config intact. If you want to clean it completely and erase the config to start fresh you can do a distclean.

```
$ CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- ARCH=arm make distclean
```

You'll notice I left the cross_compile and arch variables in there so that clean knows what to remove. It probably uses wildcards to clean, but better safe than sorry.

4.9. Final Thoughts

It's pretty simple to compile a kernel and knowing this will help you to stay current. OpenEmbedded builds a kernel as well but this source tree used here is much better.

Be careful what change in the config. Changing too many things at once without really knowing what you're doing might cause your kernel to not work as you want. If you just feel like experimenting, that's fine, but if you're in a production environment and just want something that works, be careful what you do.

```
#!/bin/bash
date >build.log
cd mini2440
CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- \
  ARCH=arm make mini2440_defconfig >>../build.log
CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- \
  ARCH=arm make >>../build.log
CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- \
  ARCH=arm make uImage >>../build.log
cd ..
date >>build.log
```

Errors concerning depmode:

Create /lib/modules/(uname -r) (run depmod to see the missing directory).

Note: Use the new kernel from the Friendly Arm website 2.6.32.2 - need to update the config enable ext2 ext3 ffs
CONFIG_ARM_THUMB CONFIG_AEABI (in processor config)

linux-2.6.32.2-mini2440_20100609.tgz

Use cross.sh:

```
CROSS_COMPILE=/usr/local/arm-2008q3/bin/arm-none-linux-gnueabi- ARCH=arm make $1
```

Chapter 5. Building and Installing Openembedded

5.1. Setting up the Directory Structure

You will need to create a working directory for oe, there should be no symlinks above it. As an example:

```
/home/user/work/oe
```

Create and name this as you wish, and you will have to substitute it in the following set ups.

Now some subdirectories need to be created.

```
$ mkdir -p /home/user/work/oe/build/conf
$ mkdir /home/user/work/oe/sources
```

5.2. Install Bitbake

Install BitBake into your oe directory.

```
$ cd /home/user/work/oe
oe$ wget http://download.berlios.de/bitbake/bitbake-1.8.18.tar.gz
oe$ tar xvfz bitbake-1.8.18.tar.gz
oe$ mv bitbake-1.8.18 bitbake

(you might also add this to your .bashrc file)
$ export PATH=/home/user/work/oe/bitbake/bin:$PATH
```

5.3. Git Openembedded

Now get the build for mini2440:

```
oe$ git clone git://repo.or.cz/openembedded/mini2440.git
oe$ mkdir sources
```

This will create the sub-directory mini2440 in your oe directory.

5.4. Prepare the Config File

We need to copy the edit the Mini2440 configuration file from the mini2440 directory.

```
oe$ cp mini2440/mini2440_local_conf_example.conf build/conf/local.conf
```

Next, we need to edit the configuration file you just copied and renamed. The new file is inside your build directory in a directory called conf. Edit local.conf from there and be sure you edit the following variables.

- DL_DIR should point to your sources directory you just created.

- BBFILES should point to your mini2440 directory that was created when you downloaded openembedded with Git. Just modify that one so that it's path is correct. Don't remove the trailing part.

```
DL_DIR = "/home/user/work/oe/sources"  
BBFILES = "/home/user/work/oe/mini2440/recipes/*/*.bb"
```

You'll also need to remove the last line of the config file that says:

```
# REMOVE_THIS_LINE...
```

5.5. Set BBPATH and Kill dash

The next step is to set an environment variable called BBPATH. You could also add BBPATH to .bashrc.

```
oe$ export BBPATH=/home/user/work/oe/build:/home/user/work/oe/mini2440  
(make certain you are not using dash)  
oe$ sudo dpkg-reconfigure dash (pick NO)
```

Important

If you set BBPATH incorrectly you will get: 'ERROR: IO Error: file 'conf/bitbake.conf' not found-
when you go to build

5.6. Building Openembedded Recipes

1. Build Task Base

Now build the task base. This could take 3-5 hours, as the tool chain and every thing else is built. Let it run 20 minutes or so before you walk away.

```
/oe/build$ bitbake -v task-base
```

Important

If you set BBFILES incorrectly in local.conf you may get: 'ERROR: Nothing PROVIDES 'task-base'

2. Build base or console image

If you have 120MB or larger Flash chip you will need to edit the file mini2440/conf/machine/mini2440.conf and find the line at the end starting with EXTRA_IMAGECMD_jffs2 and change the 0x200 in it to 0x800, and change the 16KiB to 128KiB.

```
EXTRA_IMAGECMD_jffs2 = "--pad --little-endian --squash -s 0x800 -n -e 128KiB"
```

This changes how the images are made with mkfs.jffs2 ... If you have a micro2440 SDK board with 128MB NAND, your erase block size is 128k and your sector size or page size is 2048bytes (2k, or 0x800 in hex). This is only meaningful if you plan to use flash.

Now start the console image build:

```
/oe/build$ bitbake -v console-image or base-image?  
/oe/build$ bitbake -v mini2440-image
```

All the good stuff you just spent hours building is in `oe/build/tmp`.

3. Creating a Bootable SD Card

First, you need to have uboot installed. I used `u-boot-nand16k.bin` on my 1G flash and it works.

Now, you need to format your SD Card, and to do this you will use Linux. When you insert the card Ubuntu will mount it. I umount the mount and remount it because, well because that what I do and it seems to work. For this part, I find it most convient to get a root prompt for the partitioning and formatting.

Use `fdisk` or `cfdisk` to create three partitions. Create the partitions then format them.

(replace the device name with yours)

- Partition 1: Swap Space 512M `mkswap /dev/sdb1`
- Partition 2: Boot Partition 20MB `mkfs -t ext2 /dev/sdb2` Copy uImage here
- Partition 3: Root Filesystem - The remainder of the free space, untar the root file system here

In `oe/build/tmp/deploy/glibc/images/mini2440` you will find `uImage-mini2440.bin` - copy it to the second partition as `uImage`.

You will also find `Angstrom-console-image-glibc-ipk-2009.X-stable-mini2440.rootfs.tar.gz` I will refer to this as `rootfs.tar.gz`

```
$ tar xzvfo rootfs.tar.gz -C /path/to/mounted 3rd partition
```

Now in uboot we need to set up for booting from the SDCARD:

```
MINI2440# setenv bootcmd mmcinit \; ext2load mmc 0:2 0x31000000 uImage \; bootm 0x31000000
MINI2440# saveenv
```

```
MINI2440# setenv bootargs console=ttySAC0,115200 mini2440=1tb rootfstype=ext3 root=/dev/mmcblk0p3 rw rootwait
MINI2440# saveenv
```

Unmount the card and move it to the mini2440, and boot from NAND.

The first boot will take a bit longer.

4. Now that its booting

You may wonder what now, before I build and install QT. Well guess what, you have a package manager on board, `opkg`.

First, you need an interconnect. Really the whole point of this board is the fact that Linux provides network services. Best idea is to just plug it into your router. Sould DHCP just fine.

So now that you are connected, lets install some things. Before we can, we need to tell `opkg` to get a list of stuff. Now that you have the LAN connect, you can just SSH into the board. If you are on windows, grab Putty and use that.

4.1. opkg - The installer

Get the a list of all the stuff you can get - you may get an error on some of the files.

```
# opkg update
```

Then list all the stuff:

```
# opkg list
# opkg list >stuff
```

Now you have a list of everything in the file called stuff, so we can grep that file to find things we may want

```
# grep 'nano' stuff
nano - 2.0.7-r0.1 - GNU nano (Nano's ANOther editor, or Not ANOther editor)
nano-dbg - 2.0.7-r0.1 - GNU nano (Nano's ANOther editor, or Not ANOther editor)
nano-dev - 2.0.7-r0.1 - GNU nano (Nano's ANOther editor, or Not ANOther editor)
nano-doc - 2.0.7-r0.1 - GNU nano (Nano's ANOther editor, or Not ANOther editor)
```

So, lets install nano:

```
# opkg install nano
```

4.2. The sftp server

It's pretty handy to have an ftp server, but these days you need to have a secure one. So lets install an sftp server it and start it.

```
# opkg install openssh-sftp-server
# /usr/libexec/sftp-server &
```

4.3. Web server

A web server is already installed with Busybox

Create a directory say /www and put a sample index.html

To start the web server:

```
# /usr/sbin/httpd -h /www &
```

Use ifconfig to find the ip of the board and go there in your browser.

4.4. Picocom

```
tar xfvz picocom-1.6.tar.gz
cd picocom-1.6.tar.gz
user@ubuntu:~/picocom/picocom-1.6$ export CC='/usr/loca/armv4t/arm-angstrom-linux-gnueabi/bin/gcc' ; make
```

Chapter 6. Compiling Qt and Tslib

You will need to install these packages first:

```
$ sudo apt-get install libtool
$ sudo apt-get install autoconf
```

6.1. Set up the toolchain

We will use the toolchain from Openenbedded that we just built - there seems to be certain issues with the CodySourcery tool chain, so we will copy the just built tool chain:

```
$ sudo cp -R /home/user/work/oe/build/tmp/cross/* /usr/local/
$ export PATH=/usr/local/armv4t/bin:$PATH
```

```
./configure --host=arm-angstrom-linux-gnueabi --prefix=/home/jdonelson/usr/local/tslib --enable-static --enable-shared
```

6.2. Build TSLIB

"Tslib is an abstraction layer for touchscreen panel events, as well as a filter stack for the manipulation of those events. It was created by Russell King, of arm.linux.org.uk. Examples of implemented filters include jitter smoothing and the calibration transform. Tslib is generally used on embedded devices to provide a common user space interface to touchscreen functionality. It is supported by Kdrive (aka TinyX) and OPIE as well as being used on a number of commercial Linux devices including the Nokia 770. " From tslib.berlios.de

We will use git to get the source for tslib, build then set it up for qt to build against the libs. Bill recommends installing to /usr/local/tslib because that is where it will end up on the target. I don't because for one thing it is not a great idea to install code built for arm on your x86 box and there are permission issues with this localtion.

Appendix A. Formatting the SD card

A.1. Mount the SD card

Insert the card into an SD card reader. Ubuntu 10.x will automount it.

Use `sudo fdisk -l` to find the device. Note that there is a base device and then partition devices. We need to use the base device.

Important

Make sure you are using the SD card NOT your hard drive!!

```
Disk /dev/sdb: 1015 MB, 1015808000 bytes
32 heads, 61 sectors/track, 1016 cylinders
Units = cylinders of 1952 * 512 = 999424 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x6f20736b
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdb1		1	128	124897+	83	Linux
/dev/sdb2	*	129	148	19520	83	Linux
/dev/sdb3		149	1016	847168	83	Linux

The base device is `/dev/sdb` so we need to :

```
$ sudo cfdisk /dev/sdb
```

A.2. Partition the card

We need three partitions. Start by deleting all the current partitions.

- A swap partition 512MB
- A boot partition 20MB EXT2 uImage goes here
- The rest of the card EXT3, the root file system goes here

Use `cfdisk` to delete all the partitions and create new ones.

After it is all set up, eject the card and remount it.

A.3. Format the card

```
sudo mkfs -t ext2 /dev/sdb2
```

```
sudo mkfs -t ext3 /dev/sdb3 (this takes a while sometime)
```