OSK5912 Newbie Guide

Matthew Percival

27th September 2005

Contents

1 The Usual Business

If you see anything wrong or missing in this document, please let me know so that I can update it. Additional areas where people trip-up are also welcome, if they are included with a solution and a means by which to verify this.

2 Introduction

This guide is for people new to working with Embedded Linux, specifically with the OMAP5912 Starter Kit (OSK). People who have past experience with Embedded Linux will probably find this too basic for them: it is written by a newbie for other newbies just starting out. You could see this as a case of one who has just recently stumbled through the wilderness for the first time passing on what they learned to those who are just about to enter. This guide will make the assumption that you have no prior experience, and will focus only on the easiest way to do tasks: this is not necessarily the best way, but we shall leave the complicated ways for those who know what they are doing. Building everything from the ground-up would probably be the best way to ensure you have an environment perfectly suited to your task, but this guide shall focus on starting with preprepared items, when available, to ensure things are easy to learn.

Your OSK should come with a basic system preloaded: U-Boot 1.1.1, and 'MontaVista Linux' with a 2.4 kernel. If you plan to continue working with MontaVista, this guide is not for you: they have already done all the hard work for you, so this is unnecessary. I shall describe using a non-corporate system, put together with freely available components. If you have a board with nothing installed, skip ahead to Section 9 where you will find steps for getting your board back to its factory set up.

I will also make the assumption that your development environment will also be GNU/Linux, and that you have some (very) basic knowledge of working with this Operating System. I shall not be making any large assumptions about prior knowledge, but if you have never used it before, you may need to read a beginner's guide on that topic first.

I shall include in this guide warnings in regard to areas I had trouble; if a comment seems a little strange, please do not look down on me!

3 First Steps

Assuming you have the factory system already sitting on your board, we shall first take a look at that, so you can become familiar with accessing it via a terminal. If you are using Windows, you will need to use the program 'HyperTerminal' (available under Start > Accessories > Communications); for GNU/Linux the most common choice of terminal program is 'minicom'. HyperTerminal will work fine as is, but minicom needs a little configuring first. After you run it, press 'Ctrl-A O', which will open the Options Menu. Select 'Modem and Dialing' and ensure that 'Init string' (A) is clear. Now you need to set either program so that it uses /dev/ttyS0 (minicom) or COM1 (HyperTerminal) at 115,200 baud, 8Bps, no parity, 1 stop bit, and no flow control[1].

Once these settings are correct, put a serial cable between the OSK and your PC, and plug the power into the OSK. You should now be briefly greeted by Das U-Boot, before seeing the Linux kernel start up. Before long you will receive a login screen, and will be able to try out a few commands, if you so desire.

As things currently are, no matter what you plan to do, the board is fairly useless to you like this: you now need to start setting things up! You could compile programs and send them straight across like this (assuming you are using MontaVista), but I shall now begin explaining how to set up your own custom setup, before going onto compiling and running programs on your OSK.

[1] Spectrum Digital, Inc., *OMAP5912 Starter Kit (OSK5912) User Guide*.

4 Toolchain

The gcc toolchain is something you will become most familiar with very quickly: you will need this to compile C/C++ programs you intend to use on the OSK. You can make your own toolchain, but I shall begin with describing using a precompiled one. Precompiled toolchains can be found all around the place, but the one I shall use for the purposes of this guide came from 'ftp://ftp.handhelds.org/' (currently 'ftp://ftp.handhelds.org/projects/toolchain/arm-linux-gcc-3.4.1.tar.bz2').

TIP: There are a lot of different toolchains around, so make sure you use one that fits your needs. Some are designed for different architectures, and there are many older ones around too. At the time of writing, you would want to look for a 3.4.x toolchain. 4.0.x toolchains are beginning to become more available too, however, they still not as common. I am currently using a 4.0.1 toolchain, built with gcc 4.0.1, binutils 2.15, glibc 2.3.5 and Linux 2.6.11.3 headers.

Assuming you have you have the toolchain above, you will not need to do much to set it up:

```
# cd /
# tar -xjvf ${DOWLOADDIR}/arm-linux-gcc-3.4.1.tar.bz2
```

It will take a moment to unpack your toolchain, but when completed you will find it sitting in '/urs/local/arm/3.4.1/' — that is pretty well all there is to it! To use your toolchain, you will either need to add it to your PATHs in your .bashrc file, or before each session of using it send:

```
$ export PATH=/usr/local/arm/3.4.1/bin/:$PATH
```

This is of cause assuming you use bash: if you use another shell, please adapt this to whatever is appropriate to your environment[1].

If you wish to make your own toolchain, there are plenty of detailed resources available on this topic — I would recommend "Building Embedded Linux Systems"[2]. I shall just briefly go over the essentials, and provide some resources I found useful.

[1] Texas Instruments Inc., *'Building Linux for the OMAP5912™Development Platform'*.
[2] Karim Yaghmour, *Building Embedded Linux Systems*, O'Reilly & Associates, Inc., 2003

For your arm-linux toolchain, you will need the source to the following packages:

- GCC
- glibc
- binutils
- Linux

Some people may also add GDB, and in some situations you may use a different version of GCC for the bootstrap compiler and the final crosscompiler. For some people, alternatives to these packages (eg uclibc) may be preferable, however, these shall not be covered in this document.

However, it should be noted that not just any combination of these will compile: this is where things can potentially get fiddly. You are free to try combinations on your own, however, this can potentially be quite time consuming, and it may be a good idea to check if someone else has tried them first. The linux-arm-toolchain mailing list at http://lists.arm.linux.org.uk/mailman/listinfo/linux-arm-toolchain may prove useful, as may the buildlogs for the crosstool script. The latter, for the latest version of crosstool, may be found at http://www.kegel.com/-crosstool/crosstool-0.37/buildlogs/index.html — this site list the success or failure of several builds, however, they do not list if the toolchain works: only whether is compiles or not.

Once you have identified a combination of either known-good versions, or versions you wish to trial, the basic process begins by downloading the sources for the programs you are using, unpacking them, and applying any necessary patches. From here you will need to configure the kernel and use the kernel headers to begin propagating the crosscompiler's include/ directory. Next comes building arm-linux versions of binultils, to create the better part of your bin/ directory, before creating the bootstrap compiler. Depending on your hardware, this could take quite a considerable amount of time to build. The bootstrap compiler allows you to compile glibc, which provides the bulk of the contend for your crosscompiler's lib/ directory. Finally, you build the full compiler, which may include compilers for C, C++ and other languages, before undertaking a number of housekeeping tasks.[1]

[1] Yaghmour, 2003

That description is awfully brief, and somewhat vague, but is merely intended to give you a general idea of what is involved in this process. If you want further information on the specifics, I would strongly recommend reading "Building Embedded Linux Systems"[1], or using the crosstool script — available at http://-www.kegel.com/crosstool/ to automate the process. Because most of the steps are quite straight-forward, they are easily automated, and require little — if any — human interaction, even without a script, so crosstool could prove a valuable option.

Because of the large amount of compiling that goes on in building a toolchain, this process will take several hours, even on a relatively powerful system. If you are not following a known-good combination of tools, you may also find yourself spending a lot of time only for the compilation to fail at some point, or even succeed in compiling, only to find that something else fails down the track. The latter problem may seem like a downside to building your own toolchain, but it is actually one of the main advantages: if something does go wrong, you can do something about it! You may only need to patch and recompile, or change the version of a package, however, if you use a precompiled package, you do not have this option.

[1] Yaghmour, 2003

5 Das U-Boot

Das U-Boot is the first thing you will see when you start up your OSK. If you press a key when prompted, you will be able to use U-Boot, rather than booting into Linux. From here, you can upgrade U-Boot or your kernel, or flash on a new root filesystem; you can also change environment variables, such as the kernel's boot arguments. The latest version can be found at SourceForge (http://sourceforge.net/projects/u-boot), however a precompiled (1.1.1) binary can be downloaded from 'http://linux.omap.com/pub/bootloader/osk/osk-u-boot.bin'.

If you wish to compile your own version of U-Boot, it is a fairly simple operation. First, unpack it:

```
# cd /usr/src/
# tar -xjvf ${DOWLOADDIR}/u-boot-1.1.2.tar.bz2
```

Then, once your PATH is set up (see Section 4):

```
# make distclean
# make omap5912osk_config
# make
```

Assuming everything is fine, you'll now have a 'u-boot.bin' sitting in your directory.

TIP: If you receive an absolute mass of errors reading "cc1: error: invalid option short-load-bytes", it is safe to edit 'cpu/arm926ejs/config.mk' and remove the 'mshort-load-bytes' option.

TIP: If you receive a collection of errors ending with "relocation truncated to fit: R_ARM_PLT32 __div0", it is due to a bug in your toolchain. If you have compiled from source, I shall provide a patch for binutils that I received from Richard Woodruff:

```
————————- from CodeSourcery ————————————
— bfd/elf32-arm.h.orig 2004-04-22 22:11:15.000000000 -0400
+++ bfd/elf32-arm.h 2004-04-22 22:28:37.000000000 -0400
@@ -2229,6 +2229,8  elf32_arm_relocate_section (output_bfd,
case R_ARM_PC24:
case R_ARM_ABS32:
case R_ARM_THM_PC22:
+ case R_ARM_PLT32:
```

```
+
if (info->shared
&& (
(!info->symbolic && h->dynindx != -1)
@@ -2262,11 +2264,6  elf32_arm_relocate_section (output_bfd,
relocation = 0;
break;
- case R_ARM_PLT32:
- if (h->plt.offset != (bfd_vma)-1)
- relocation = 0;
- break;
-
default:
if (unresolved_reloc)
_bfd_error_handler
```
——————————————— end patch ———————————-

This should solve that problem.

Once you have a binary (either self-compiled or pre-built), your next step is to upload it to your OSK. I shall assume you already have an existing version of U-Boot on the board: if not, Section 9 has instructions on how to install it onto a virgin system. There are several ways you can send U-Boot to the board, but I shall detail tftp as I found it the fastest and easiest way to do this. These instructions, as well as instructions for Kermit transfer, can also be found at 'http://-linux.omap.com/pub/documentation/U-Boot for the OMAP16xx SDP.pdf'.

You will first need to install a tftp server package: there are several available and, as always, each distribution has its own way to install these. Please follow the normal procedure for your distribution. Once it is installed, the first step is to configure your tftp server. Firstly, edit '/etc/xinet.d/tftp' (on some systems it will be init.d rather than xinet.d) and remove the line 'Disable = yes'. You will then need to ensure that the following (or something very similar) is present in the file:

```
service tftp
{
socket_type = dgram
protocol = udp
wait = yes
user = root
server = /usr/sbin/in.tftpd
```

```
server_args = -s /tftpboot
disable = yes
per_source = 11
cps = 100 2
}
```

Finally, ensure that the permissions of '/tftpboot/' are at least 'drwxr-xr-x'. Note that you may have to make this directory yourself, depending on your distribution.

You are now ready to upload your new U-Boot. Load your OSK as described in Section 3 and, at the prompt, press a key to use U-Boot. We now just have to set up U-Boot for tftp. Firstly, after ensuring it is connected via the Ethernet port to your system (it need not be direct: you can go via a network), we need to set up the environmental variables. Firstly, we need to tell it the MAC Address on the board (you'll find this printed near the serial port):

```
# setenv ethaddr 00:0E:99:XX:XX:XX
```

Next we set the IP details. If you have DHCP configured, you can do this with a simple 'dhcp' command. Otherwise:

```
# setenv ipaddr ${OSK_IP}
# setenv serverip ${PC_IP}
# setenv netmask ${NETMASK}
# setenv gatewayip ${GATEWAY_IP}
```

If you do not know these details, you will need to get them through 'ifconfig' and 'netstat -rn' commands. Once everything is set, we just need to save this information:

```
# saveenv
```

When you are ready, send your 'u-boot.bin' to '/tftpboot/', then return to your terminal. At the prompt here, we download the binary with:

```
# tftpboot 0x10000000 u-boot.bin
```

When the download is complete, it will report the size of the file in hexadecimal: note this number! We then need to remove the old U-Boot, then put the new one one top:

```
# protect off 1:0
# erase 1:0
# cp.b 0x10000000 0x0 xxxx
```

Where xxxxx is the hexadecimal number you noted earlier. You can now reboot your OSK, and the new U-Boot should load[1]. We're now ready for the next step!

[1]Texas Instruments Inc., *'U-Boot for the OMAP 16xx GSM/GPRS Development Platform'*.

6 Kernel

Embedded Linux really begins with getting Linux itself together. You can download the latest kernel from ‘http://www.kernel.org/’ and if you visit ‘http://www.muru.com/linux/omap/’ you will be able to get the necessary patch to use this kernel on an OMAP system. At the time of writing, the latest available patch is for the 2.6.13.4 kernel.

Once you have these files, you will need to extract and patch the kernel:

```
# cd /usr/src/
# tar -xjvf ${DOWLOADDIR}/linux-${VERSION}.tar.bz2
# cp ${DOWLOADDIR}/patch-${VERSION}-omap1.bz2 .
# bunzip2 patch-${VERSION}-omap1.bz2
# cd linux-${VERSION}/
# cat ../patch-${VERSION}-omap1 | patch -p1
```

Now your kernel is ready for you to prepare. The standard steps to set up your kernel, after ensuring your PATHs are set correctly, are as follows:

```
# make clean
# make omap_osk_5912_defconfig
# make menuconfig
```

The last line could also be ‘make xconfig’, simply ‘emacs .config’, or whatever you are most comfortable with. I will make the assumption you have some basic Linux knowledge here, so I shall leave it to your discretion to modify the kernel to suit your needs. The current settings will be enough to simply run on the board though.

TIP: Make sure you have the following in ‘.config’ before continuing:

```
CONFIG_NFS_FS=y
CONFIG_NFS_V3=y
CONFIG_ROOT_NFS=y
```

You will need them for the next section!

Now we just need to compile the kernel:

```
# make
```

Once this is completed, you will need to do a few finishing operations on the kernel to make it ready for use[1]:

```
# arm-linux-objcopy -O binary -R .note -R .comment -S arch/arm/-
    boot/compressed/vmlinux linux.bin
# gzip -9 linux.bin
# ${U-BOOTDIR}/tools/mkimage -A arm -O linux -T kernel -C gzip
    -a 0x10c08000 -e 0x10c08000 -n 'Linux Kernel Image' -d linux.-
    bin.gz uImage.cc
```

TIP: If you did not compile U-Boot earlier, you may find yourself in a little trouble with that last step. It is worth your while going back to that step, even if just to compile the 'tools/' directory.

If you now send your 'uImage.cc' file to the '/tftpboot/' directory, we are ready to put the new kernel on your board. Load up your OSK as before, and go into U-Boot. Once there:

```
# tftpboot 0x10000000 uImage.cc
```

Again, note the hexadecimal size of this, as you will need it soon. Then we erase the old kernel and put the new one on:

```
# erase 1:8-15
# cp.b 0x10000000 0x100000 xxxxx
```

Where, as before, xxxxx refers to the hexadecimal number you noted earlier.

TIP: If you receive an error like this:

```
Copy to Flash...not erased at xxxxxxxx (xxxx)
Flash not Erased
```

You will need to erase more than the 8-15 range (kudos to Prakash for this information). Finally, we set some boot parameters for the kernel:

```
# setenv bootargs console=ttyS0,115200n8 noinitrd ip=${OSK_IP}:
${PC_IP}:${GATEWAY_IP}:${NETMASK}:osk:eth0:off root=
/dev/nfs rw nfsroot=${PC_IP}:/data/rootfs2.6,nolock mem=32M
# saveenv
```

[1]Texas Instruments Inc., *'Building Linux for the OMAP5912™Development Platform'*.

Note that some of those settings are in anticipation of the next step, and as such (unless you have already done the next step) things are not going to work quite yet. You are now ready to start the kernel:

```
# bootm 0x100000
```

TIP: If your OSK hangs after decompressing the kernel, that is due to a bug in the factory U-Boot. If you upgrade your U-Boot, this problem with be solved. Another way to get around this error is to make a couple of small changes to your kernel source. Firstly, add to ‘arch/arm/boot/compressed/Makefile’:

```
ifeq ($(CONFIG_ARCH_OMAP),y)
OBJS += head-omap.o
endif
```

Then add the following to ‘arch/arm/boot/compressed/head-omap.S’:

```
#ifdef CONFIG_MACH_OMAP_OSK
/* support for booting without u-boot */
mov r7, #(MACH_TYPE_OMAP_OSK &  0xf)
orr r7, r7, #(MACH_TYPE_OMAP_OSK & 0xf)
#endif
```

Recompile and repeat the process from above to replace the old kernel with this patched one. Now your problem should be solved (thanks Vineet Tuli for linking me to this, originally shared by Kyungmin Park).

7 Root Filesystem

Once you have your kernel running, the last step to have your board ready for use is to prepare the root filesystem. A root filesystem can be prepared in a very short period of time, but a basic one can be downloaded from 'http://linux.omap.com/-pub/filesystem/rootfsosk.tar.bz2' too. The essentials, beyond a handful of directories, mainly come down to a small selection of dynamic libraries, and the ever-useful tool 'BusyBox'. There are also a handful of configuration files and the like, but it is mostly trivial.

I will not cover building your own from scratch, as there are already many documents covering what the essentials of a root filesystem are; if you want to read more, I would suggest you read "Building Embedded Linux Systems"[1]. Assuming you have downloaded the prebuilt one mentioned above, we just need to unpack it to a directory of its own:

```
# mkdir /data
# cd /data/
# tar -xjvf ${DOWNLOADDIR}/rootfsosk.tar.bz2
```

Congratulations, the essential directory structure and a well-organized version of BusyBox is all ready for you to use. If this is enough for your purposes, you can skip ahead to the NFS part of this section, but I shall go over a few other things first. Note also that the configuration for BusyBox used here may not satisfy your needs, so please do check this out first.

Firstly, the '/lib/' directory of this filesystem is awfully bare, so you will need to stock it up with libraries from your toolchain (/usr/local/arm/3.4.1/arm-linux/-lib/). I will leave which libraries you want to copy across up to you: it'll all depend on what you need later on. You could copy *.so and *.so.[*0-9] across, if you wanted to, but that would almost certainly be using up a lot more space than you need to.

Secondly, as I said, the version of BusyBox in that file is a little dated. You can get the latest version from 'http://www.busybox.net/'. As always:

```
# cd /usr/src/
# tar -xjvf ${DOWNLOADDIR}/busybox-${VERSION}.tar.bz2
```

Again, so long as you have your PATH set, we can use a nice little menu to configure BusyBox to suit our needs:

[1] Yaghmour, 2003

```
# make menuconfig
```

If you are unsure as to what to include, the existing programs on the filesystem you downloaded will serve as a great guide, and you can add any extras you believe you may need at this point. You should also configure it to use the cross-compiler, and install to '/data/rootfs2.6/', then install like so:

```
# make TARGET_ARCH=arm all install
```

Once you are happy with the basic setup of your root filesystem (additions are easily made at any time), it is time to set up your PC to share it via NFS. Firstly, you will need to install whatever NFS packages are appropriate to your distribution - I shall pass over this step, as it varies too much between distributions. Once you have the basic packages installed, we need to do a little setting up. Firstly, edit '/etc/default/portmap' to remove '-i 127.0.0.1' from 'ARGS=' (this step is not required on all distros), then restart portmap:

```
# /etc/init.d/portmap restart
```

Now create '/etc/exports' and add a line like '/data/rootfs2.6 *(rw,no_root_squash,no_all_squash,sync)'. Then we restart a couple of daemons:

```
# /etc/init.d/networking restart
# /etc/init.d/nfs-kernel-server restart
```

When we now boot up the OSK, with the Ethernet connection in place (as when we did tftp), we should leave U-Boot and let it go straight to the kernel. Once Linux finished loading, we will either have a login screen (login with root, no password) or go straight to a prompt - this depends on your BusyBox configuration. You are now ready to use your customized OSK as you please!

TIP: If you get an error like 'VFS: No root yet, retrying to mount root on NFS (unknown-block(0,255))' and cannot boot any further, you have made a mistake somewhere along the line. Either your NFS shares are not correctly configured (you can test this with a second PC, if available) or your kernel was not compiled with the correct options. Note that the necessary options are not in the default settings, but are mentioned in Section 6.

At some point you will want to be able to use this system independently: there is only so much you can do with the OSK relying on a host system. This is where you will need to make a filesystem image and flash that to your board. There are a lot of different filesystems you can use for your image (eg cramfs,

jffs), but I shall describe using JFFS2. Before you even begin to worry about the filesystem format, however, you will need to get things ready for 'Memory Technology Devices' (MTD henceforth).

Firstly, you will need to install the MTD utilities onto your host system. As always, follow the usual procedure for installing new components for your distribution. This will be mainly important for getting the mkfs.jffs2 program for later.

The last bit of preparatory work we need to undertake is to get MTD support into the kernel. Ensure you have the following in your '.config':

```
CONFIG_MTD=y
CONFIG_MTD_CONCAT=y
CONFIG_MTD_PARTITIONS=y
CONFIG_MTD_REDBOOT_PARTS=y
CONFIG_MTD_REDBOOT_DIRECTORY_BLOCK=-1
CONFIG_MTD_CMDLINE_PARTS=y
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLOCK=y
CONFIG_MTD_OMAP_NOR=y
```

If you do not, they can be found in the 'Device Drivers > Memory Technology Devices (MTD)' section. Add them, recompile, and install your updated kernel as per the steps earlier.

With that, you are now ready to load a filesystem image. It is quite simple to make this image file:

```
# mkfs.jffs2 -p -l -e 0x20000 -n -v -r /data/rootfs2.6/ -o filesys.jffs2
```

It may be useful to explain this command. -r ${DIRECTORY} selects the source for your image - in this case, the root filesystem that had been used for the NFS mount. -o {$IMAGE_NAME} is where you name the image file this program outputs. -p is used to pad the end of the image with '0xFF', -n stops it from writing 'CLEANMARKER' nodes, and -e {$BLOCK_SIZE} specifies the erase block size. Using those three will prevent a whole host of errors from appearing later on! Finally, move this image to your '/tftpboot/' directory.

We now have to upload this image to the OSK. If you return to the U-Boot prompt, as you had to upload the kernel earlier. From here, we follow a process you are likely becoming familiar with:

```
# tftpboot 0x10000000 filesys.jffs2
```

Note the hexadecimal size of this, as you will need it again. Now we erase the entire filesystem block on the flash device:

```
# erase 1:128-255
# cp.b 0x10000000 0x1000000 xxxxxx
```

Wher, as usual, xxxxxx refers to the hexadecimal number you noted earlier.

TIP: If you do not erase the entire block, you will get some nasty errors later on. It it a lot simpler to do it now than to come across those problems later and have to repeat this process.

The final step is to update the boot arguments for the kernel. This may vary somewhat for your set up, so if these settings do not work for you, just read ahead and I shall explain how to find the settings you need. You can mostly use the same boot arguments as before, but rather than ‘root=/dev/nfs rw nfsroot=${PC_-IP}:/data/rootfs2.6,nolock’ we now need ‘root=/dev/mtdblock3 rootfstype=jffs2’. If you make that simple substitution, everything should work fine. Boot your kernel as before (with the ‘bootm’ command, or a simple reboot), and you should soon be running an entirely independent OSK.

TIP: If your get a kernel panic and an error about not being able to mount mtdblock3, then your filesystem may be located on a different mtdbock. When your kernel boots, look for something similar to this:

```
Creating 4 MTD partitions on "omapflash.0":
0x00000000-0x00020000 : "bootloader"
0x00020000-0x00040000 : "params"
0x00040000-0x00240000 : "kernel"
0x00240000-0x02000000 : "filesystem"
```

Counting from 0, the filesystem is number 3 in this list, but it may be 4, 6, or something altogether different for you. If you change the mtdblockx value in your boot arguments to the number where your filesystem is located, then you should be able to boot the system.

TIP: If you get errors like: ‘CLEANMARKER node found at 0x00010000, not first node in block’, you forgot to add -n to your mkfs.jffs2 command.

TIP: If you are getting errors like: ‘jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001e0008: 0x5000 id’, you probably used the wrong value of -e in your mkfs.jffs2 command. To get the correct erase block size, simply use the following command from the OSK:

```
# cat /proc/mtd
```

The number listed in the ‘erasesize’ column is what you require. If you rebuild the image with this value, re-flash it, and boot Linux again, you should find the errors have gone.

8 Programs

We now have our system up and running: the final step is to write our own programs for it, and run them on the system. Things are pretty straight forward at this point, so I shall not go into any great detail. Just to show you how things are working, we will make a simple ‘Hello World’ program in C:

```
#include <stdio.h>
int main (void)
{
printf (“Hello World!”);
return 1;
}
```

Once you have this code in ‘helloworld.c’, and have ensured your PATHs are correct as always, we just need to compile it with the cross-compiler:

```
$ arm-linux-gcc -Wall -O2 helloworld.c -o helloworld
```

If we now send ‘helloworld’ to ‘/data/rootfs2.6/bin/’, we should see it appear on our OSK, assuming it is currently set up (come now, how do you expect to run something on it if it’s not set up!). If we simply head over to our terminal and run our program we should see the output. This basic concept stands up for any future programs you write. I make the assumption that you already know programming fundamentals if you plan to write anything for your OSK, so I shall not go into any detail on adapting this to larger projects.

TIP: If you cannot seem to run your program, yet can see it there, you probably don’t have the loader or appropriate libraries available in ‘/lib/’ on your root filesystem. If you copy ‘ld-2.3.2.so’, ‘ld-linux.so.[*0-9]’, ‘libc.*.so’ and ’libc.so.[*0-9]’ from your cross-compiler’s library directory, you should now be able to run your ‘Hello World!’ program (credit to Manoj Kotnala for this tip). Please do note that some of these are symbolic links, and need to be copied with the ‘-d’ option If you have a different program, and those libraries alone were not enough, try this:

```
$ arm-linux-readelf -a ${PROGRAM} | grep “Shared library”
```

It will report to you what you need to ensure is present in your ‘/lib’ directory.

9 Woops!

Made a mistake, and now nothing works? Have a virgin system that needs to be installed for the first time? It is actually a fairly quick and easy process to get your OSK back to factory settings. You just need to get a few things together first: 'Flash Recovery Utility' (http://tree.celinuxforum.org/CelfPubWiki/FlashRecoveryUtility?action=AttachFile&do=get&target=OSK5912FlashRecoveryUtility.zip), a kernel, and a good filesystem. The latter two are both available on the CD that came with your OSK. A U-Boot binary is also required. One comes with the Flash Recovery Utility, but it is far from ideal: if you have another binary available (see Section 5) you can replace the default one with this early on.

Flash Recovery Utility is a fairly straight-forward Windows application, so I shall not describe it in detail. Instructions for using it come with the compressed file I listed above, though they can also be found at 'http://tree.celinuxforum.org/CelfPubWiki/FlashRecoveryUtility'. If you want to use a different U-Boot binary to the default one provided, simply rename the 'u-bootbinOSK' file to something else, and replace it with your own version of U-Boot.

Once you have re-flashed your drive and have U-Boot on, we're ready to get things going. If you configure your system for tftp (see Section 5) and then boot your OSK as normal, we are ready to finish the job. Firstly, we shall restore the factory environmental variables:

```
# setenv bootcmd bootm 0x100000
# setenv bootdelay 10
# setenv bootfile uImage
# setenv bootargs console=ttyS0,115200n8 noinitrd rw ip=off root=/dev/mtdblock4 mem=30M
# saveenv
```

Now we are ready to finish the job. Copy the factory kernel and filesystem from your CD to the '/tftpboot/' directory, then set up your OSK for tftp (see Section 5 again). We shall first download and install the kernel:

```
# tftpboot 0x10000000 uImage
# erase 1:8-16
# cp.b 0x10000000 0x100000 b4f20
```

And, finally, the root filesystem:

```
# tftpboot 0x10000000 factory-filesys.jffs2
# erase 1:128-255
# cp.b 0x10000000 0x1000000 e34b14
```

Now your system is back to its factory settings. You can either use as-is (with MontaVista), or you may like to go back to the start and follow the process of setting up your OSK by yourself[1].

NOTE: The original version of the Flash Recovery Utility available at the above link does not work with revision D boards. If you are not sure which revision your board is, check the top of the board. Below the large "OMAP5912 OSK" label is a serial number ending in 'REV X', where X is the hand-written revision of your board. The product was recently licensed by Software Design Solutions, Inc. (http://www.softwaredesignsolutions.com/) who have released a new version capable of working with both revision C and D boards. Alternatively, you can use OMAP Flash Loader, which only works with revision D boards. It can be found at the same place as the old version of Flash Recovery Utility. The OMAP Flash Loader is a Linux program, however, so is not available to Windows users.

NOTE: An undocumented item with both utilities is that you must not have a serial cable connected when trying to use them. In full boot, it will look on the serial connection if there is one, only only failing that will it try the USB connection.

[1] Spectrum Digital, Inc., *'Restoring the Factory Configuration'*.

10 Credits

As this guide was written from one newbie to another, it would obviously laughable for me to claim that I knew this all myself from the outset! Most things have been through the help of people on the http://linux.omap.com/ mailing list. Without their assistance, I would probably still be trying to come to grips with the basics. I have already credited those whose assistance has come directly into the guide earlier, but I shall list here the other sources I used in getting this far.

Karim Yaghmour, *Building Embedded Linux Systems*,
O'Reilly & Associates, Inc., 2003.

CE Linux, 'Flash Recovery Utility',
<http://tree.celinuxforum.org/CelfPubWiki/FlashRecoveryUtility>,
(accessed 23 March 2005).

Spectrum Digital, Inc., *OMAP5912 Starter Kit (OSK5912) User Guide*.

Spectrum Digital, Inc., 'Restoring the Factory Configuration',
<http://omap.spectrumdigital.com/osk5912/factoryconfig/>, 2004
(accessed 25 March 2005).

Texas Instruments Inc., 'Building Linux for the Innovator Development Kit for OMAP™ Platform',
<http://focus.ti.com/lit/an/swpa011/swpa011.pdf>, (accessed 25 March 2005).

Texas Instruments Inc., 'Building Linux for the OMAP5912™Development Platform',
<http://linux.omap.com/pub/documentation/osk/omap5912osk_2.6.pdf>,
(accessed 23 March 2005).

Texas Instruments Inc., 'U-Boot for the OMAP 16xx GSM/GPRS Development Platform',
<http://linux.omap.com/pub/documentation/U-Boot for the OMAP16xx SDP.pdf>, 2004 (accessed 23 March 2005).