Hardware hacking on a budget; the Amstrad E3

Jonathan McDowell <noodles@earth.li>

June 12, 2006

Introduction

There has long been a perception that the skills and resources to work on interesting hardware projects are available only to a select group of hackers. While this may have been true in the past there are a large number of consumer devices coming to market that are based upon Linux. As a result it's becoming easier and easier for anyone with some programming experience to get involved with hardware and embedded systems programming. This paper intends to explain how to get started with such a project and use a case study involving the Amstrad E3 videophone to illustrate the various steps.

The Plan of Attack

Your first ports of call once you've got your hardware should be the manufacturer's website and a search engine. You're looking for a copy of the GPL source used on the device and any details about the components used you can find. If you can find a list of the chips used then you can start to look for their datasheets, which will help with figuring out what they do, how they might be hooked up and how to program them. If you can't find such a list then consider opening up the device and physically inspecting it. Take pictures - when you've put it all back together and realised you've forgotten to check something you'll be glad of them.

See if you can find other people working on the same device. Never underestimate the power of the community. If you can't find anyone else working on it then put up some details about what you've discovered so far. Even a basic website pulling together all the information you've gathered can help and it means the next person who starts on the same project will hopefully be able to find you and combine efforts.

The hardest part could well be finding the console; this may require some physical disassembly and soldering in order to let you connect to a serial port on the device. However this allows you to view the bootloader and kernel messages and is invaluable. If you've no previous electronics experience don't be put off by this - eBay can sometimes provide pre-assembled cables, or your device may have something that's easily accessible.

Look for similar projects. See if there's mainstream Linux support for the CPU. If not, then is there a maintained tree that does support it? You want to avoid reinventing the wheel, so any existing support you can build on will help.

Finally, once you're done you should try to feed any patches you have back to upstream.

A Case Study: The Amstrad E3

This paper intends to use the work on the Amstrad E3[1] to illustrate the various steps of hardware hacking. The E3 is a desktop videophone that connects to a normal phone line, currently available on the UK high street for around £30 (having previously been sold for £100). Amstrad subsidise the cost of the hardware by means of requiring it make a daily premium rate phone call to remain operational.

As a desktop videophone the E3 presents an attractive form factor for modification. It features a 480x320 LCD screen (unfortunately only DSTN), a camera, a handset and both an on device keypad and a pull out 'mailboard' QWERTY keyboard. A TI OMAP5910[2] dual core ARM/DSP powers the device and is paired with 32MB of NAND flash and 32MB SDRAM. Of particular benefit to the would be modifier is the provision of a serial port on the back. This allows us to begin work without even having to disassemble the device.

Get the source

Amstrad make the offer for the kernel source used on the E3 as part of the initial introductory email the customer receives upon registering the device. They charge £25 (payable only by cheque; there is no known way to obtain the source online from Amstrad even with a card payment) to cover their administration and distribution costs. In return for this fee they send a CDR containing a 2.4.18 based tree. At first glance this appears to be the right thing; it has mentions of the Amstrad Delta (the codename for the E3) as well config options for some of the various hardware used.

Closer examination revealed that the source was in fact not what was used to build the kernel the E3 ships with. In addition to the cosmetic issue that the source is version 2.4.18_mvl30-E3 while the boot log of an E3 shows 2.4.18_mvl30-ams-delta the E3 boot log clearly shows a dfdblk/MFS-DFD driver compiled into the kernel, providing access to the NAND flash where the root filesystem lives. There is no sign of this driver in the supplied source.

(This is a GPL violation on Amstrad's part. Although they were contacted in January 2005 about the issue and refunded the author's £25 payment there has been no resolution at the time of writing. It is not expected there ever will be and an alternative Free driver for the NAND has already been accepted into the linux-mtd git tree.)

The lack of complete source is not unusual. In most cases there are binary modules handling proprietary parts of the device (e.g. wireless drivers). If you're lucky there will already be projects working on a Free driver (such as the bcm43xx driver for Broadcom wireless chipsets). If not then it may be possible to obtain a datasheet that provides enough information to write a driver. And either way whatever you can get of the source used will help, even if there are missing sections.

[1]http://www.amstrad.com/products/emailers/e3.html
[2]http://focus.ti.com/docs/prod/folders/print/omap5910.html

Examine your device

The next task was to work out what was driving the E3. The device was disassembled and the PCB examined. A TI OMAP 5910 (ARM9 + DSP dual core), 128KB NOR flash, 32MB NAND flash, 32MB SDRAM, a hardware Conexant modem and a Phillips smartcard reader were all observed and model numbers noted.

Datasheets for these parts were then searched for. In particular TI have a rich range of documentation for the OMAP cores available for download, which made working on the device a lot easier than it could have been.

See who else is out there

Although the author had discussed the E3 with some friends before purchasing his, he was unaware of anyone else who had such a device. His investigations into other groups who might have been working on the same project led him to Ralph Corderoy's website[3] about the E-mailer Plus, an earlier Amstrad desktop smartphone. While the hardware in the Plus and E3 differs quite considerably (the Plus has a grayscale screen, an earlier ARM core that doesn't have an MMU, as well as less memory) it seemed likely that Amstrad would have reused some of the same techniques from the Plus on the E3. This eventually turned out to be the case, especially in the area of the primary boot loader, which was instrumental in providing a way of easily getting code onto the device.

Find a console

If your device is running Linux the chances are high it has a serial console somewhere (if it doesn't have an alternative display device that might be used). Most embedded CPUs have at least one serial port on chip these days and these are thus frequently configured up as the Linux console. If you're lucky the serial console will provide you access to a bootloader and shell. It'll also let you see the boot messages which can provide some insight into how things are setup.

For example, many of the consumer routers that run Linux have a bootloader which will allow you to tftp a new kernel to RAM, allowing easy testing without having to repeatedly reflash firmware.

The E3 serial port presents as a 3.5mm jack socket (i.e. a headphone style socket) on the rear. This requires only a simple 3 wire cable to connect it up to a normal PC serial port, which make it very easy to get up and running.

However once the serial console was discovered it didn't provide any easy way to affect the boot process, or even end up at a shell after booting. It did at least provide some information about the steps involved in the E3 boot which was useful though. And when the process of interacting with the boot loader was figured out it has become the method used to replace the Amstrad supplied firmware.

[3]http://inputplus.co.uk/ralph/emailer/

Share and Enjoy

A basic webpage[4] describing the E3 and the parts used in it, as well as linking to the publicly available datasheets and kernel source Amstrad had provided was setup. In addition a mailing list[5] was created, originally for discussion of just the E3 but quickly expanded to include discussion about earlier models.

The list discussion originally revolved around getting the Plus to run user supplied code; both in terms of the protocol for talking to PBL and then also about the format of the code blocks that were executed from flash. Both of these turned out to be applicable to the E3, with Amstrad having used the same concepts with only slight modifications.

There was also a long period of stagnation with progress in achieving anything, once shell access had been obtained. At this point it was possible to obtain shell access to the device with the 2.4.18 Amstrad kernel, but no easy way to store files to the flash device. And even at this stage (mid 2005) 2.4.18 was somewhat ancient.

All this changed in early 2006 when some new people became interested in the E3 and found the list. In particular Mark Underwood took the knowledge that had already been gained and managed to get u-boot[6] and 2.6.15-omap1 booting. This was real progress and catalysed the project into further E3 development. If the list hadn't existed (with archives that Mark was able to look through) it is unlikely Mark would have been able to get going so quickly, nor that it would have been found by as many people.

More recently we've had an IRC channel, which has proved useful for some general chat about approaches but isn't quite as useful for newcomers as the list archives.

When in doubt, disassemble

Sometimes you just get stuck. There's no datasheet for the chip you want to talk to, there's no source for the driver that's currently in use, and you're stuck with an ancient/buggy kernel as a result.

While not necessarily as fully featured as commercially available tools, objdump is quite able to provide a disassembly of many architectures (and indeed, if the platform is running Linux it's highly unlikely objdump won't be able to deal with it). The thought of having to wade through the disassembly of 64k+ of code probably doesn't sound appealing, but it's not as bad as you might think. In particular, all you're trying to do is understand what's going on; not code in assembly yourself.

For the E3 the initial application of this was for the primary boot loader. Someone had desoldered the flash ROM and read the contents, providing an image of PBL that could be worked on. A similar process had been used for the Emailer Plus. The E3 disassembly indicated that the bootloader communication protocol was very similar to that used in the Plus. In addition it provided details about the checks used to verify the blocks stored in flash (whose format had again been decoded by those working on the Plus, but had slightly different parameters due to the hardware differences).

[4]http://www.earth.li/ noodles/hardware-e3.html
[5]http://www.earth.li/cgi-bin/mailman/listinfo/e3-hacking
[6]http://u-boot.sourceforge.net/

Later on disassembly was used to work out how to talk to the NAND flash. A datasheet was available for the part, but it wasn't known how it was connected to the processor. Amstrad had left a module of the driver on the E3 filesystem, despite it being compiled into the kernel. Checking the part of the initialisation routine where the chip was identified and comparing to the datasheet was enough to find out which I/O pins the chip control lines were connected to. Armed with this information writing a driver within the Linux MTD framework was comparatively easy.

It's important to note that disassembly can be illegal in some jurisdiction, whereas others allow it for the purposes of interoperability. 'Clean room' techniques are sometimes adopted for complicated instances, where one person/team will perform the disassembly and document the interfaces/hardware and then another will use the documentation to write a new clean driver. As they'll never have seen the original code this helps avoid claims of copyright infringement. So far for the E3 this hasn't been necessary; initial disassembly was purely to work out how to talk to PBL, and then to work out the control lines for the NAND, rather than anything more complicated in terms of writing drivers.

Don't reinvent the wheel

It's important not to waste your energy on already solved problems. If you can find an existing project to base your work on then it's more likely you'll be able to push changes upstream.

For the E3 the linux-omap[7] project was the logical place to look, as it already supported the processor used. Support was already present in mainline 2.6 kernels, but there was additional support and drivers present in the -omap tree. For example, they already had a framebuffer driver for the OMAP LCD driver, which meant that adding support for the E3's screen became a lot less work.

Also for userland both buildroot[8] and OpenEmbedded[9] have been used in the E3 development. These provide an easy way to get a cross compiling toolchain and initial root filesystem image built.

If it had been required to add support for the OMAP core to Linux, or produce a cross building environment it would have taken a lot more effort. Being able to use these existing projects certainly made life a lot easier.

Push back to upstream

This goes hand in hand with not reinventing the wheel. If you end up needing to produce patches in order to make your device work, do try to push them back to upstream. As well as meaning that others can get hold of your work a lot more easily it's also provides you with several benefits.

Firstly, it provides more eyes to look at your code, from people who know the area your code touches and who might be able to provide some constructive criticism about it, or confirm it's on the right track.

Secondly it gives you less baggage to carry around; rather than having to rebase your patches against each new release if your changes are part of upstream

[7]http://linux.omap.com/mailman/listinfo/linux-omap-open-source
[8]http://buildroot.uclibc.org/
[9]http://www.openembedded.org/

it's much more likely they'll continue to work. Too often hardware manufacturers choose a particular kernel version and stick with it, rather than trying to get their changes in mainline.

At the time of writing base support for the E3 has been accepted into the mainline kernel. Patches for the LCD have been submitted to the linux-omap tree (one of Mark's has already been accepted), while patches for the NAND and LEDs have already been accepted into subsystem trees. David Ford's work on OpenEmbedded has been committed to their monotone repo. It is hoped that further patches for u-boot and OPIE will be ready for submission soon, with the ultimate aim being that no out of tree patches are required in order to support the device.

Conclusion

The author hopes that the case study outlined in this paper has showed showed that hardware hacking is an easier task to get involved in than you may have thought. Even without having a lot of hardware experience it can be possible to provide a useful contribution to a project similar to the E3, or even get one started yourself. There are more and more Linux powered devices available on the high street providing lots of opportunities for anyone who wants to work on this often neglected area.

Further links

There are a number of hardware projects already out there that may provide the reader with some further useful reading material or ideas about how to go about hardware hacking.

- OpenWRT[10]

  A Linux distribution for wireless routers. Originally started for the Linksys WRT range it now supports a wide range of consumer kit.

- NSLU2[11]

  This project concentrates on various network attached storage devices.

- Linksys WMA11B[12]

  Andrew Wild's work on the Linksys WMA11B, a wireless media player.

- OpenEZX[13]

  The OpenEZX project tries to gather information about the Linux-based Motorola EZX phone platform (mainly the A780, E680 and E680i phones).

- Xanadux[14]

  Porting Linux to HTC smartphones (eg Orange M5000, O2 XDA).

[10]http://openwrt.org/
[11]http://www.nslu2-linux.org/wiki/
[12]http://www-jcsu.jesus.cam.ac.uk/ acw43/projects/wma11b/
[13]http://openezx.org/
[14]http://wiki.xda-developers.com/index.php?pagename=Xanadux