An ARM based web kiosk system

Vincent Sanders
Daniel Silverstone

Revision History

Revision 1.00	5th March 2009 Initial Release.	VRS, DJAS

Table of Contents

This article describes how to construct a Linux-based embedded web kiosk for an ARM platform.

1. Introduction

This is the fourth article in a series demonstrating the fundamental aspects of constructing embedded systems.

This article covers the targeting of the web kiosk system from the previous article to a specific ARM based hardware platform.

This document, and indeed the whole series of articles, assumes a basic understanding of a Linux-based operating system. While discussing concepts and general approaches these concepts are demonstrated with extensive practical examples. All the practical examples are based upon a Debian- or Ubuntu-based distribution.

2. Scaling the design

The previous articles have not required any additional hardware, indeed by using the QEMU emulator we have been able to keep all the examples confined to the realm of software.

It is rare however for an embedded system to have no limits on its hardware resources. In fact, it is common for a project brief to severely limit the available hardware platform, generally through cost and size, but increasingly through power usage. The cost and size limits are self evident however the power limitation bears a closer examination.

Embedded systems like the web kiosk example are generally in a fixed location which presents the opportunity to have a full mains power connection. In this situation why might power consumption be an issue?

The primary emerging reason is that running electrical cabling for power to these systems limits their location and may have high cost implications. If, instead, the systems can be powered entirely from a single data cable, using low voltages, they become much more flexible.

A second consideration which has gained greater significance in recent times is that of environmental impact. Appliance applications where the unit is powered for extended periods of time, and must always be available, by their very nature have a

large environmental impact. A PC might consume 100W on average, over a day that builds up to 2.4KWh or some 876KWh per year, which equates to roughly 400kg of coal or 377kg of carbon dioxide released into the atmosphere (conversion factors from DEFRA [http://www.defra.gov.uk/environment/business/envrp/conversion-factors.htm]). If we produce a system which can operate from just 12W, we reduce the yearly figure to just 105kWh or 45kg of carbon dioxide. A side effect of this is, obviously, reducing the electricity bill by over 80% which makes economic sense.

Figure 1. Web Kiosks

The obvious power supply method for a networked system is to use Ethernet cabling employing the IEEE 802.3 Power-over-Ethernet (PoE) standard. This however introduces a maximum available power of 12 Watts (24W for the much less common IEEE 802.3at standard). Running a standard PC and LCD monitor from 12W is simply not possible with current available technologies.

If the project brief calls for an integrated display we may plausibly use a directly connected LCD panel in a fashion similar to a laptop. By using a directly connected panel, the refresh rate may be lowered significantly, which could reduce power draw. A direct connection also removes the need for power sapping line drivers which are required for standard monitors to connect over long cables. This approach might let us reduce the display power requirements to around 10W. This only leaves 2W budget for the computer, clearly a PC of any description cannot operate within such a power budget and an alternative solution must be sought.

The most common low-power computing platforms are currently based upon ARM System On Chip (SoC) devices. These devices operate at a few hundred megahertz instead of the gigahertz speeds found in PCs and consume a fraction of the power. There are numerous SoCs available, in a huge array of hardware solutions, and selecting the correct one for a given application is something to be discussed in a future article. One important factor in selecting a suitable system for developing a Linux solution is it must have kernel support. A useful reference for what systems are supported in current ARM Linux kernels is the KAutobuild [http://armlinux.simtec.co.uk/kautobuild/] project.

For the purposes of this article a Simtec Electronics thin client system [http://www.simtec.co.uk/products/BBD20EUROA/intro.html] with a 400MHz Samsung SoC module [http://www.simtec.co.uk/products/IM2440D20/] has been selected. This system provides reasonable CPU performance coupled with a GPU capable of driving large screens. This hardware would be capable of being directly connected to an LCD panel as discussed previously, and has sufficient power supply flexibility to have PoE functionality added if required.

An important idea which should be highlighted here is that, for many embedded applications, suitable hardware is often already available. Embedded projects often start with a faulty premise that they will need bespoke hardware solutions specifically for their application.

Often existing products can be modified or extended to produce completely acceptable results at a fraction of the cost. It is especially important to be realistic about the volumes of a product. A product which will have small production volumes will by definition have a higher per unit cost when the Non Recurring Engineering (NRE) cost is amortised over all the units produced. Thus, although purchasing and customizing an existing product may have a relatively high per unit cost, the NRE will be much lower.

To make this clearer let us use a real example. For a hundred units the thin client system selected costs GBP200 per system (approximately USD300) or GBP20,000 pounds total. A fully bespoke design from Simtec Electronics for a system of equivalent complexity would cost approximately GBP40,000 and have a manufactured per unit cost around the GBP150 mark for small runs. For a hundred unit run this means the bespoke approach has a unit cost of GBP550, indeed the GBP200 per unit cost is not arrived at until almost 300 units have been produced. This is a trivial example and includes no NRE cost for required software, such as a boot-loader and a Linux kernel port, which the pre-built system already includes, and no budget for contingencies or specification changes.

The practical result of this is that, for fewer than a thousand units, the “customized off-the-shelf” approach nearly always makes economic sense.

3. Converting the example

The method we have employed so far is to use a shell script to copy binaries from a host system and generate an initramfs cpio archive to boot. Just because we have changed architecture there is absolutely no reason not to continue in this way.

Because the board has no local storage attached (there is provision for such; we are simply not using it) the instructions for bootstrapping the system [http://www.simtec.co.uk/products/SWDEBIAN/files/debootstrap-article.pdf] with a Debian armel installation accessed via NFS were followed. This produced a system we can develop on in *exactly* the same way as our x86 host previously. Because Debian provide support for numerous architectures, this approach can be used just about anywhere that a Linux kernel, with the appropriate options, can be started.

The actions to build the browser were identical to those in the previous article. Indeed because no graphical environment was installed, the NetSurf browser could be run directly from the NFS mounted system.

The configuration script [http://www.simtec.co.uk/products/SWLINUX/files/mkbusyfs/webkiosk-arm.sh] for the ARM web kiosk system is very similar to the original. The main difference is in the number of modules required. It is drastically reduced because a specifically-configured pre-built kernel was used. This kernel has the drivers built-in for the limited peripheral set found on the development boards made by Simtec Electronics.

The `webkiosk-arm.gz` and the kernel were placed on a TFTP server and the ABLE boot-loader was used to start the system.

```
> (tftpboot)vmlinuz-2.6.26-simtec1-s3c24xx-eabi initrd=(tftpboot)webkiosk-arm.gz \
  root=/dev/ram0 console=ttySAC0
```

The system started and displayed the web browser as expected. The drawback to the host-based approach has become evident at this point however. Compiling the web browser on the ARM box took some seventeen minutes, preparing the cpio took another four minutes, shutting down the NFS system and booting the output took another six minutes.

From these timings it can be seen the edit, build and test cycle using this approach is over half an hour which, as we learned in the second article of the series, will have a large impact on the developer's productivity. We might make some improvement on that time by not having to rebuild the entire browser every time and by using a faster boot medium than TFTP (the ABLE boot-loader supports HTTP, for example).

Because the system was targeted at a Simtec Electronics integrated module it is possible to run the system on other base-boards in the range. To test this the image was started on a DePicture [http://www.simtec.co.uk/products/BBD20EUROU/] touchscreen tablet. The system started as expected and was able to navigate the web as normal with the touchscreen.

Figure 2. DePicture system showing the NetSurf welcome page.

4. Improving the process

The primary issue with the approach illustrated here, is that of development speed. The hardware system was selected for its ability to meet our project requirements, these are different to those necessary to construct the system.

The hardware selected traded CPU performance for a reduction in power consumption. The processor in PCs in general use at this time has around ten times the CPU power and five times the memory performance of the ARM SoC.

We could of course simply build on a more powerful ARM host and deploy the output on the target. This approach has the merit that it is straightforward to implement and the substantial drawback that a compatible host may not be available. Even if a compatible system is available they must be purchased for each developer which adds cost and may not shorten the development cycle by an amount large enough to justify the expense.

A commonly-used solution to this issue is to harness the power of a full PC system to generate output for the target. We have already used the QEMU emulator to test our output images, it can also emulate targets other than an x86 PC. Although some of the performance of the host processor will be lost to the emulation, the resulting system will generally still perform better than the native ARM hardware. This gives us the desired build performance without the expense of purchasing additional physical hardware. It also means we do not have to reboot the build system to test the results.

One issue worth mentioning here is that the more powerful the host PC the faster the emulation will be and the shorter the development cycle. If the host PC is not up to the task this approach might actually result in the opposite of the desired result and slow development.

The bootstrapping [http://www.simtec.co.uk/products/SWDEBIAN/files/debootstrap-article.pdf] article outlines how to create a disc image suitable for use with the emulator. A Debian Lenny system for the armel architecture should be prepared and started with QEMU and the system used to generate images.

The only small caveat is that the chosen target hardware system must be supported by QEMU. Unfortunately it is somewhat challenging to get new hardware support accepted into the QEMU project. At this time the Simtec Electronics boards are not supported in the pre-packaged QEMU, patches and sources necessary to create a QEMU that supports Simtec Electronics boards are available but must be compiled and installed separately.

5. What's next?

This fourth article outlines some of the reasons to use hardware more suited to a set of requirements. It demonstrates the relative ease with which a host based build approach can be used to generate a working system for a small ARM system. Finally it shows some of the drawbacks of this approach and how they might be addressed.

In the next article we shall examine more issues surrounding the completion of a project and examine an alternative approch to building systems.

6. About the authors

Vincent Sanders	Vincent is the senior software engineer at Simtec Electronics and has extensive experience in the computer industry. He has worked on projects from large fault tolerant systems through accounting software to right down to programmable logic systems. He is an active developer for numerous open source projects including the Linux kernel and is also a Debian developer.
Daniel Silverstone	Daniel is a software engineer at Simtec Electronics and has experience in architecting robust systems. He develops software for a large number of open source projects, contributes to the Linux kernel and is both an Ubuntu and Debian developer.
Simtec Electronics [http://www.simtec.co.uk]	Simtec is a full solutions provider with a proven track record of helping clients with all aspects of a project, from initial concept and design through to manufacturing finished product. With 20 years in the industry, and producing ARM CPU modules since 1992, Simtec's wide experience in embedded systems and the Linux kernel provide a strong base on which to build custom hardware and software solutions, from the smallest of USB devices to the largest complex Linux systems. Simtec's custom-off-the-shelf design service, utilising a range of pre-designed modules of various functions, allows for rapid design and prototype turnaround, reducing time-to-market. Simtec also provide a full software development consultancy with an extensive range of products from boot loaders to full Linux based operating system environments and a range of development boards showcasing Simtec's modular designs.