Linux embeds itself

Successive iterations of the original Linux kernel have made it increasingly relevant to embedded applications, from the rugged industrial environment to consumer products, as Lamberto Ascoli of Future Electronics Europe explains.

Linux hasn't always been favoured for embedded applications. Prior to the release of the 2.4 version, the Linux kernel really didn't do itself any favours in the embedded space: primarily because of its size, real-time performance, determinism and scalability. It was more geared towards its historical stomping ground of the PC server domain. As such most engineers either developed their own embedded operating system (OS), or used a commercial OS such as Neutrino from QNX Software Systems or a real time operating system (RTOS) such as Wind River's VxWorks, or Microsoft's MS-DOS or Windows CE for creating a desktop PC-like environment.

Successive iterations of the original Linux kernel culminating in the latest 2.6 release, however, have significantly improved things to the extent that many product manufacturers now consider Linux a fully viable embedded OS.

Indeed the world's first Linux-powered smartphone – the Motorola A760 – has just been launched. Despite using Linux for the first time, this high performance device has apparently made zero performance tradeoffs and also exploits Java technology to offer full multimedia PDA functionality.

It joins an ever growing list of embedded Linux-driven products (see www.linuxdevices.com) including PDAs, handheld computers, wrist watches, robots, audio video entertainment systems, PC tablets, security systems (e.g. Ethernet cameras), industrial controllers (operator terminals, network gateways, maintenance controllers), and IT network devices (firewalls, thin clients and wireless hubs).

The main reason for this success is that because embedded Linux is both open source and royalty-free it helps simplify and reduce the cost of complex silicon design cycles and accelerate time to market. There is also a good range of embedded Linux software development tools from major vendors such as Metrowerks, and a large number of ready-to-go silicon devices and design platforms from manufacturers such as Motorola, AMD and QuickLogic.

Inevitably, there are pitfalls to be avoided: the open source nature of Linux alone means care has to be taken about which software is incorporated into a design (as its source and quality may be of unknown or dubious quality).

In addition, if you alter the kernel (risky so not common) or develop a new application (very common) you are obliged to make your changes public under the rules of Linux's governing GNU Public Licence (GPL). Copyrighting can also be problematic.

This article will present the benefits of using Linux in embedded applications, examine a representative range of development tools and silicon, and highlight the common challenges to be overcome. Because Future Electronics supplies all of these products, it can give an impartial overview of this rapidly developing field and can draw upon a relatively large amount of in-field experience.

The historical context

Developing a microprocessor-based embedded design was traditionally a hardware challenge. Because processors typically comprised CPU with no peripherals the development process involved selecting a processor based on both performance and the availability of development systems, emulators and compilers. To this would be added various functionality blocks such as timers, serial or parallel ports, and interrupt controllers. Finally – after spending time struggling with timing charts and hardware simulations – a final design would emerge.

Firmware or software was an afterthought in terms of design priority and would generally be developed in-house: very rarely was any kind of OS needed or desired due to the enhanced performance and cost that would be passed onto the original processor selection.

Modern embedded processors have little in common with their predecessors of even two or three years ago. They not only include vastly larger amounts of processing power, they come equipped as standard with all the peripherals that could reasonably be required within an embedded application.

The peripheral list on the latest Motorola PowerPC MPC5200 – see figure 1 _ and the AMD Alchemy series Au1100 – figures 2 – for instance, includes serials, timers, DMA channels, memory, CAN and LCD controllers, USB host, and high speed Ethernet (to name but a few). The AMD Geode family is also worth mentioning as although it has fewer peripherals than the Alchemy series, its x86 compatible means it can host Windows XP embedded too (in addition to all the embedded OS's mentioned above).

Although Linux has been successfully ported to processors that haven't a memory management unit (MMU) (for example the very stable and well-known µCLinux running on the Motorola Coldfire family and other cores), if the full Linux kernel is required a PowerPC, ARM or MIPS core is usually preferential.

For even greater design flexibility, there are field programmable gate arrays (FPGAs) offered by vendors such as QuickLogic in the form of its QuickMIPS family. QuickMIPS is also an antifuse technology device, which means it impossible to reverse engineer and thus helps protect IP – particularly relevant given the increased pirating concerns associated with offshore low cost manufacturing regions.

Furthermore, for all of these silicon products an emulator is not needed, because the debugging resources are on chip, and evaluation boards are supplied from each manufacturer with core schematics, PCB Gerber files and bill of materials data to hasten along the development cycle.

Although these technological developments are welcome news to any embedded design engineer, in reality the design challenge has not disappeared: it has simply moved out of the hardware and into the software.

Standards interoperability can be an issue with Ethernet or USB ports, for ex-

ample, and will demand that the firmware and software complies with what will typically be a very large (thousand page) technical document that many embedded designers will not have had recourse to open before. And this may just be the start of a very long list that can include compliance with equally rigid Internet and communications protocols such as HTTP, SNMP, FTP, SSL, and POP3. Such requirements demand an embedded OS and the development of the right software components within it.

Many commercial embedded OS's have been available for years, each with their own application scope.

If the application is a handheld or portable computing device that will be used to gather information in a retail store or warehouse and feed it direct to a PC network, then Microsoft Windows CE .NET may be a good choice because of the ease with which it can be ported to the MS Windows desktop OS.

Alternatively, if the application is of a safety-critical or high reliability nature – e.g. controllers for some part of a railway network, power station or chemical process plant – then reliability proven and dynamically upgradeable code such as Neutrino from QNX Software Systems may be considered the best choice. And for genuinely hard real-time performance such as high speed and precision motor control or defence applications, then the Wind River VxWorks RTOS might be the most suitable OS.

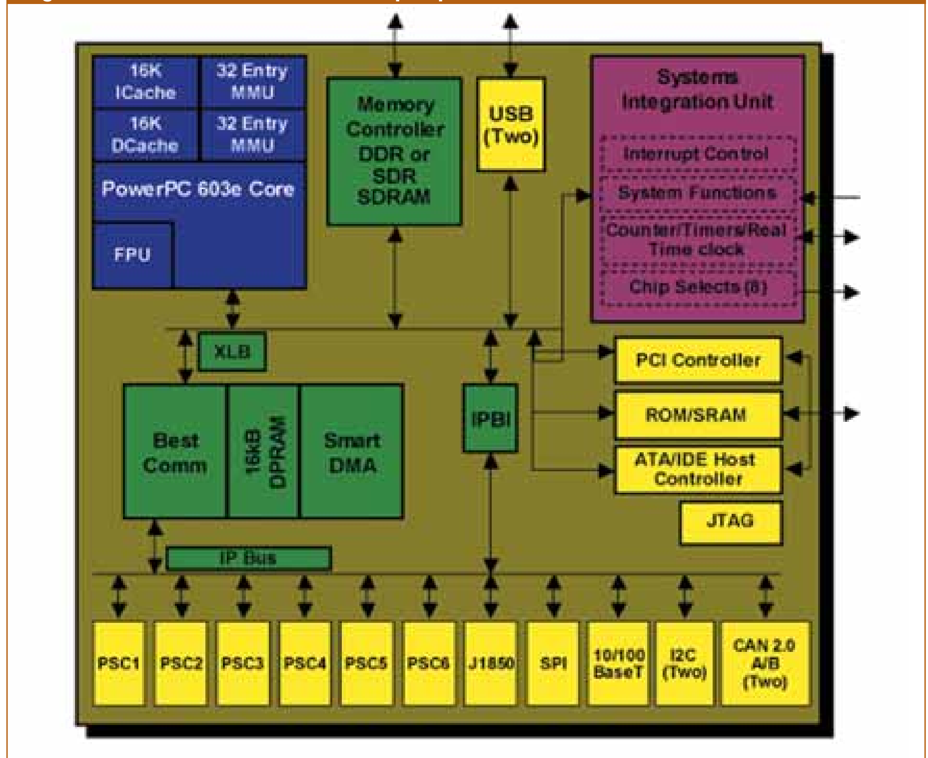
Finally, the DIY embedded OS route continues to be popular, especially for companies with a large development team capable of counteracting the higher NRE costs and time-to-market delays, or with a large amount of legacy software available.

Yet for the majority of embedded developers that don't need hard real-time performance, perhaps lack the resources required to develop their own OS with a full set of features, and can accept having to put any application modifications into the public domain, then embedded Linux could be ideal.

Nonetheless, there are a number of important considerations that need to be given careful consideration when deciding to use Linux.

Linux code is best developed from within a Linux OS and this may mean having to get to grips with how to install, configure and use a Linux desktop PC OS – such as Red Hat or Suse Linux – and an inevitable start-up delay. A dual boot configuration with Windows and Linux may be the best solution to have the new OS maintaining compatibility with an existing OS.

Fig 1: Motorola PowerPC MPC5200peripheral list



For those not ready to change, several software packages exist to create a Linux environment on Windows PCs, for example the free Cygwin (www.cygwin.com). Whichever route is chosen, time should be scheduled for testing and exploring the new platform.

One of the most widespread requirements in industrial applications is real-time functionality. Linux is not strictly an RTOS. The main original design objective behind the Linux kernel was throughout not real-time and predictability. As such, the kernel has not historically been pre-emptible and while the processor executes kernel code, no other process or event can pre-empt kernel execution.

This situation has since been rectified to a large degree – particular in version 2.4 onwards – by a number of patch files designed to provide real-time to Linux. However, it does still mean that if an application demands high levels of real-time accuracy (i.e. a latency of a few microseconds or less), Linux may not be applicable. This is particularly true if network stacks and complex device drivers are used. For most industrial applications, however, this level of real-time performance is not required.

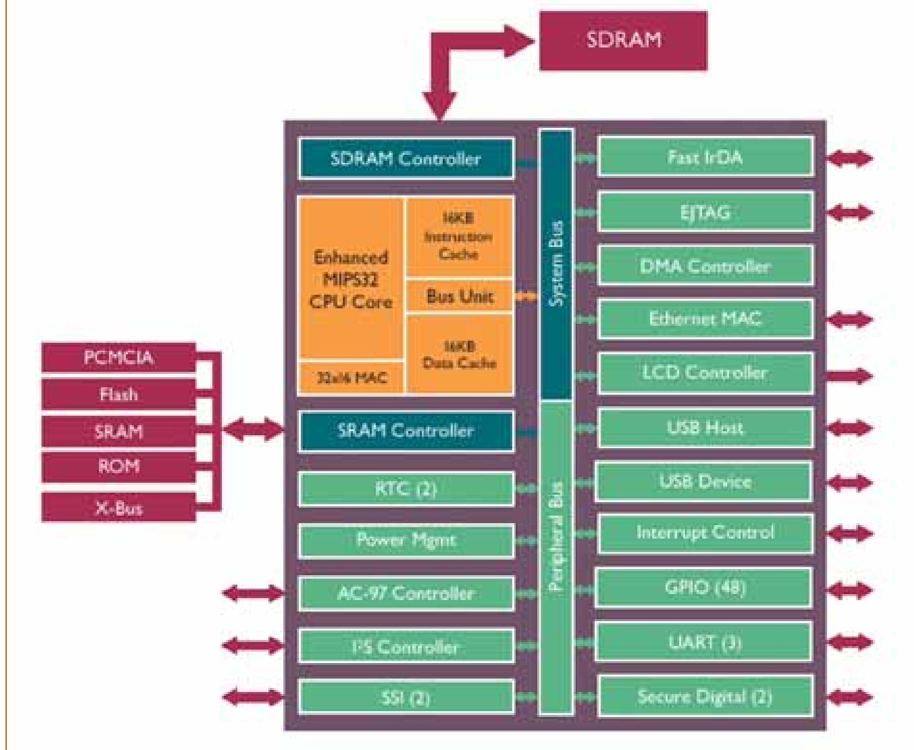
One of the most clever Kernel-tweaking patch techniques for increasing Linux's real time performance to a few tens of microseconds latency comes from Metrowerks that contributed to the development of a dedicated patch and integrated it into its own OS. Indeed a detailed technical white paper [1] describing two methods is freely available.

Alternatively, a pre-existing Linux Real-Time Application Interface (RTAI) that allows developers to write applications with strict timing constraints is freely available [2].

Linux is best regarded as a turnkey platform rather than turnkey solution. The addition of a USB host interface illustrates the point. The USB host controller hardware is built into most high end processors or can be added using an external component (such as the Cypress Semiconductor CY7C67300 part). In the Linux Board Support Package (BSP) a hardware driver can easily be found. The low level Host Controller Interface software is included in Linux, so, when the peripheral is connected, a software identification process starts to reveal the identity of the peripheral itself. Then, the OS should automatically launch the device specific driver. If the device is a standard mouse or keyboard it is likely that a pre-existing Linux driver can be found. If it is a custom developed USB device, a suitable driver should be added to the Linux image by the developer, for example using the flash memory file system.

Similar considerations apply to the graphical environment. If an LCD screen is

Fig 2: AMD Alchemy series Au1100 peripheral list



used in the end application for the user interface, an application framework for code reuse, higher level programming and standard graphic primitives availability will typically be required. When Linux is booted, it doesn't load any GUI (Graphical User Interface) by default. When you see the console with the "#" prompt, it means the LCD controller hardware driver is correctly installed and setup in the BSP, so the system is ready to 'go graphical', but you have to select the framework needed.

For PC-based Linux, 'Xfree86' is a very common choice, but it is too large for an embedded environment. MicroWindows was a standard choice a few years ago because of its small memory footprint, but other frameworks (PowerParts for Linux, Qt/Embedded) are becoming more popular for enhanced performance. Regardless of choice, the framework has to be added to the Linux image, and loaded on start-up or demand.

Development platforms

The first task in creating a Linux OS for a specific application is to select a development platform. A ready-to-use image will typically be available for most evaluation boards, but it should be modified as soon as the specific custom hardware is ready. Two approaches can be used to accomplish the OS creation task: collecting and merging free pieces of software, or using a complete suite already set up and ready to use by a third party software company. In both cases, the software 'building blocks' are the same, so the final quality may be identical, but the time required to assemble the module and setup the OS can very dramatically.

The well-known (GNU's Not Unix) GNU tool chain is freely available on the Internet (GNU is a free Unix-like software OS that was launched in 1984). Compiler, linker and command line debugger are ready to download and install. A graphical interface for debugging may also be desirable, and several free options can be found to lie on top of the GNU debugger. With the free tools, one can save money but there are some risks.

Nothing to pay also means no automated support tools, an extended setup time to get the process up and running, and virtually no technical support. Although there are large user groups on the Internet that can offer a solution to most kind of issues, it's hardly a professional or guaranteed method for yielding the desired response. As such an unexpected problem may keep a project stuck for days: it's unlikely, but it is a risk. As such the 'DIY' approach may not be the best option for novice users.

Although tools supplied by software developer companies are not free, their level of sophistication and support generally reduces start-up time and improves overall productivity to a level that makes them well worth the initial investment.

The designer gets support from professionals aware of all the details of the tool in all the phases, from installation to development. Agreement can be further extended to special support such as the development of customer specific drivers or patches. Metrowerks Platform Creation Suite (PCS) for Linux is a good example of the value software technology can supply to embedded designers (3). It also allows developers to develop legitimate but uniquely customised Linux code without having to publish it.

Metrowerks PCS for Linux is a framework of several GUI tools giving the user the capability to tailor and extend Linux to their own specific application requirement.

Starting from what's available in the 'software cloud' that is the mass of open source software available to everybody. A few examples include 'Target Wizard' – a tool for selecting and building OS components with the click of a mouse; 'Debian Binary Import Tool' that makes it easy to import an embedded Linux image pre-compiled applications, selected from a directory of nearly 4000 items and available for the desktop in the Debian repository (www.debian.org); and the 'GPL Compliance Tool' that alerts to potential intellectual property issues, reporting compliance to open source license types.

Once the OS is up and running, the Metrowerks Codewarrior Development Studio is an ideal companion tool for application developers, giving access to project management, compiling and debugging facilities from a single integrated environment. Available for both Linux and Windows hosts, it can adapt to any IT corporate platform.

There is little doubt that the popularity and success of embedded Linux will continue to grow. In particular for consumer embedded applications, the accelerated time to market made possible by being able to access existing code is a major commercial benefit.

Although the open source nature of embedded Linux does take some getting used to for most manufacturers, it does also mean that product differentiation is left to the value of the actual end application. Like embedded Linux itself, this is a new but improved commercial paradigm that should help ensure longer term success.

References

1.www.linuxdevices.com/news/NS5134111490.html
2. www.aero.polimi.it/~rtai/
3. www.metrowerks.com/MW/Develop/Embedded/Linux