by Bob Japenga (USA)

Getting Started with Embedded Linux (Part 2)

Choosing a Platform for Your Embedded Linux System

You know when to choose Linux for an embedded design. This article details the platform types that can run embedded Linux. With this information, you can move ahead and choose the proper foundation for your embedded system.

I remember my first engineering project from my senior year in college. Our task was to design a weather satellite and my team was responsible for the control system. I remember sitting with my teammates over a set of high-level requirements and a blank sheet of paper. Four months later, our design was complete. Now, 40 years of projects later, I remember this as one of the few that started with a blank sheet of paper. Since most real projects we work on are built based on some previous work, we rarely have complete control. Almost everything I have done in my career has been accomplished by standing on the shoulders of the giants before me, seeing what others have done, and building upon that. By the way, most of us attribute this metaphor to Isaac Newton. But he was quoting the 12th century French neo-Platonist philosopher Bernard of Chartre.

What, you may ask, does this have to do with choosing a platform for hosting an embedded Linux system? As we look at the options available, I understand that most of us don't start with a blank sheet of paper. Choices will be made for you by others who will have selected the target cost of the system, the delivery schedule, or the processor architecture. I hope to help you make the choices that you do have control over. I'll start with two options that most of you will never have, in order to help you understand the scope of what is available with embedded Linux.

THE MOST COMPLEX OPTION

Imagine you were developing a brand new processor, starting from a completely blank sheet of paper—a new whiz-bang machine, unlike anything else ever developed. Or, perhaps you want Linux to run on an existing processor, one with no Linux support. How would you build a version of Linux for this processor? You could start by visiting the official website for the Linux kernel which is maintained by the Linux Kernel Organization (see Resources). From there, you could download the latest stable or long-term kernel. At this point, you would need to choose one or two processors that were at least somewhat like yours, look at what they did, and develop all the support code necessary to make your processor run Linux. You would be developing

SOME HELPFUL DEFINITIONS

Kernel: the software that is the bridge between hardware and the software applications, providing access to and management of the hardware and software resources
Toolchain: the software that is used to create other software, usually including a text editor, a compiler, a linker, and a debugger
Distribution: a specific package of the Linux operating system that contains the kernel, the toolchain, and the application software
Platform: the hardware for hosting an embedded Linux system

memory-management code, timers, and memory interfaces, all patterned after the Linux kernel design paradigm you see in other processors's Linux code. Eventually, you would also need a toolchain for this new processor, but that is another discussion.

Architecture	Notes
Alpha	You guessed it—Digital Equipment's PC killer
ARM	One of the most popular embedded architectures
CRIS	Code Reduced Instruction Set: CPU used in network servers (discontinued in 2011)
FR-V	Fujitsu's FR-V Microcontroller family
H8300	Renesas H8/300 Series: high-speed processors featuring powerful bit-manipulation instructions, ideally suited for real-time control applications
I386	X86 family of CPUs
IA64	The Itanium architecture of CPUs, specializing in explicit instruction-level parallelism
M32R	A 32-bit RISC architecture CPU developed by Mitsubishi for embedded systems
M68k	The Motorola 68000 CPU architecture
MIPS	The MIPS CPU (Microprocessor without Interlocked Pipeline Stages) is a RISC developed by MIPS Technologies
PPC and PPC64	Power PC 32- and 64-bit CPUs
PA-RISC	Precision-architecture RISC CPU developed by Hewlett Packard
S/390	IBM S/390 series of CPUs
Super H and Super H64	A 32/64-bit RISC architecture of Microcontrollers for embedded systems developed by Hitachi
SPARC and SPARC64	The Scalable Processor ARChitecture, developed by Sun
V850	Renesas's CPU architecture, targeted at real-time processing applications
X86-64	X86 64-bit family of CPUs

Table 1—Some of the processor families/architectures supported by Linux

THE NEXT OPTION

Just as I have rarely started with a blank sheet of paper, you will probably not be asked to put Linux on a brand-new processor family. You may, however, be called upon to put Linux on a processor that was built by the giants before you. In that case, you will just add a new processor within an existing family of processors. Table 1 provides a list of some the families currently supported by the Linux Kernel Organization. This list will give you an idea of the breadth of support Linux has in the processor community. Under each of these families, there can be scores of individual processors.

If you put Linux on a new processor, your software will be added to the list of supported processors for your family (called an architecture). As before, you would be modifying all of the key hardware interfaces, modifying the toolchain, and then building the necessary Linux applications.

GETTING TO REAL EXAMPLES

I would hazard a guess that few of you fall into either of these first two categories. More likely, you are putting Linux on a system with a processor that's already supported.

If your processor is already supported, use the Linux configuration tool provided for the kernel and select your processor, the various peripherals (such as the PHY), and all of the glue logic for your project. If these are supported, then build the kernel and the associated support applications (i.e., file system, network options, etc.), and away you go.

Hold on. What if your peripherals and glue logic are not supported? Great question. At this point, you are no longer standing on the shoulders of giants. Instead, your hardware designer is standing on your feet. If the hardware is already cast in copper, you are stuck and will need to write your own drivers, expending a lot of time and money. To simplify your task, all the peripherals and glue logic you place around the processor should preferably be selected from the list of those supported. This means you should try to be involved in the hardware design early on. Fortunately, a lot of devices are supported. Unfortunately, I have never found a definitive, up-to-date list, because it is constantly changing. My approach has been to take the Linux kernel source code I have downloaded and search the driver tree for the particular A/D, real-time clock, PHY, or I/O expander I am using and try to find out if it's supported. Just use the first few characters of the device part number and see what you find. If the device is supported, you'll need to find out whether that driver is supported by your specific processor. For this, you can use the Linux configuration tool. Sometimes, the chip vendor will say that the device is supported under Linux, it may even provide the driver. For example, Analog Devices recognized the importance of providing Linux drivers, and has provided a comprehensive list of its peripherals with Linux drivers (see Resources). I expect others will soon follow.

Thus far, I've glossed over another important part of your project. All the supporting application software needs to be compiled for your particular processor. Things like the C library (assuming you are programming in C), the file system, ntp, telnet, or a web server will all need to be built for your particular processor. I've also skipped over another important issue: your toolchain. Stay tuned for more on that in my next column.

OPENEMBEDDED OPTION

The Linux Kernel Organization approach is not for the faint of heart. OpenEmbedded is slightly better. This open-source project was created to enable embedded-system developers to create a complete Linux distribution. There is an active community of developers, just like you and me, who are building embedded systems. Unlike the Linux Kernel Organiza-

tion's kernel.org, which is used by the entire Linux community, OpenEmbedded is specifically targeted to create the kind of systems you and I work with.

OpenEmbedded supports a wide variety of architectures and processors, but there are some unsupported architectures, such as the DEC Alpha. You can choose your processor following a set of recipes and build not only the kernel, but the toolchain and the supporting Linux applications.

We have used OpenEmbedded to move a project built around a reference design (see below) to a later version of the kernel. At the end of the project, the entire package was well configured and the process was repeatable. (It reached Level 2 on the Capability Maturity Model continuum of software development, which was developed by the Software Engineering Institute at Carnegie Mellon University to help improve software reliability.)

The disadvantage of using OpenEmbedded is that there is a steep learning curve to learn how to use its tools. It can take weeks to get Linux up on your board.

REFERENCE DESIGN OPTION

If you're starting to feel dizzy, just hang in there. It's getting easier, not harder. Another option for a hardware and system designer who hopes to use Linux is to choose a reference design, which is supplied by chip or board manufacturers. My company has used reference designs from Cogent Computer Systems, Texas Instruments, and Atmel. They provide schematics, application notes, a bill of materials, and a complete Linux distribution. This approach is quick and easy. You can begin writing software on a development board, which is usually similar to your own board at the computer architecture level. You can design your board with the exact same or similar parts to those used in the reference design, and then use the same Linux distribution you used on the development board. We have had Linux running on our proprietary board within one day when we use this approach.

A disadvantage to this approach can occur if the manufacturer does not support the processor or the supporting chips. If you look at the Analog Devices webpage I referenced earlier (see Resources), you'll note that the company has indicated whether or not each device is supported. For products with long life spans and many software iterations, this is critical.

Another problem can happen when you want to add a device that is only supported in a newer kernel. For example, we wanted to add a new Wi-Fi USB device and found that it required a newer kernel or back porting the driver—both painful processes. If all of your other drivers are not supported in that newer kernel (e.g., because the chip manufacturer never put them into the standard kernel), adding that device will be painful. When this happened to us, we were severely limited in our ability to add new devices the customers demanded.

SINGLE-BOARD COMPUTER OPTION

If your budget can tolerate it, perhaps the simplest platform to build a Linux project on is a single-board computer (SBC) or a module (SOM) that comes with Linux already on it. There are literally thousands of such SBCs and SOMs. With this option, you can forget about spending any extra time getting embedded Linux to work, and benefit immediately from its power. In other words, you can get straight to what makes your company unique: your application.

Obviously, this comes at a higher recurring cost. However, I have found that it works well in two situations: when the production volume shipped is low (less than 1,000 per year) or when the time to market is critical. In one of our projects, a company needed to hit the ground running. To make money in its business model, it needed to sell more than 3,000 per year, but it was willing to take a loss on the first few thousand. So, we designed the system around a SBC and delivered the prototype within nine weeks, and a fully qualified version—one that meets the standards of Underwriters Laboratories (UL), the Federal Communications Commission (FCC), and the American National Standards Institute (ANSI)—within six months. Then, after it established market presence and its volumes skyrocketed, we redesigned the company's system around an ARM9 reference design from Atmel. All the application code now runs on both systems, despite the differences between the two processors.

Finally, another one of our customers built their system around a SBC using an X86 architecture using a compact flash for memory. In this case, you can take a standard Linux distribution—such as Ubuntu, Fedora, or Red Hat—drop it on the card, tweak the BIOS to boot from the card, and go. This customer knew nothing about Linux, but developed a successful embedded Linux application by booting from flash memory this way without our help!

ENDLESS OPTIONS

Price, schedule, and history can all affect the platform options available to you. But there are many options. There is much we haven't yet covered, and what we have covered has only been in thin slices. But then, that is what this column is all about. See you next time, when we'll talk about toolchains and licensing issues.

Bob Japenga has been designing embedded systems since 1973. In 1988, along with his best friend, he started MicroTools, which specializes in creating a variety of real-time embedded systems. With a combined embedded systems experience base of more than 200 years, they love to tackle impossible problems together. Bob has been awarded 11 patents in many areas of embedded systems and motion control. You can reach him at rjapenga@microtoolsinc.com.

RESOURCES

Analog Devices Wiki, Linux Driver, 2011, http://wiki.analog.com/resources/tools-software/linux-drivers-all?force_rev=1.

The Linux Kernel Organization, www.kernel.org

OpenEmbedded, 2011, www.openembedded.org.

SOURCE

ARM9 Reference Design
Atmel Corp. | www.atmel.com