



The eCos RTOS-

An Open Source Alternative

Step aside Linux, another open source alternative is better for embedded systems. It excels at configuration and resource management, and is more portable for design reuse. Even better, it's a COTS alternative that's less costly in the long run.

Anthony J. Massa, Technical Software Consultant
eCos

There are a wide variety of RTOSes on the market today and deciding which one to use is usually straightforward. Most designers stick to conventional evaluation metrics such as verifying that the kernel provides standard features like pre-emptable threads with multiple priority levels, a variety of clocks and counters, and multiple methods for synchronization such as mutexes, semaphores and message boxes. In addition, the typical RTOS evaluation concerns low latency interrupt and exception handling, memory management and the availability of device drivers for serial, Ethernet, flash ROM, USB and PCMCIA devices.

But these are all *conventional* metrics used in evaluating an RTOS, and most offerings meet all of these criteria without too much trouble. However, other, less obvious factors should also be considered.

For instance, how well does the RTOS and development environment handle configuration management and resource utilization? Moreover, what about the ease with which the RTOS can be ported to a different processor or the level of support provided by industry? These issues can often become a make-or-break decision on both the current

and *future* project.

And when considering open source versus proprietary RTOS choices, not all open source choices are as “free” as they appear to be. Care should be taken to balance the cost of the RTOS—in terms of royalties, up-front fees, or the opportunity cost of turning your hard-earned IP into open source for the competition. Let's examine these unconventional decision-making metrics and see how a variety of RTOSes stack up. In many embedded projects, the open source eCos operating system is the best choice.

Resources and Configuration

One of the key concerns in embedded systems deals with resources and getting the most out of these resources. Whether it is processor cycles or memory footprint, an embedded developer's job is to meet the deadlines of the system requirements while fitting the software into the space provided. Similarly, reusing one core design in multiple applications with different devices and real estate constraints becomes easier if the software is flexible enough to easily adjust to available devices; this is just another form of resource management.

As embedded systems are pushed to be smaller, faster, cheaper and more sophisticated, control over all software in the system is necessary. For example, if a particular embedded device does not

need any networking support, the network support software modules, including networking stack and device drivers, should be easily removed. This is key in an embedded environment where resource usage and real-time responsiveness are major concerns. If a software module is not being used for a particular application, it can be eliminated from the system, freeing up resources for other modules that are used.

Resource management is really an outgrowth of system-level configuration management (hardware and software). A superior configuration method aids the different development cycles for a product. As the product evolves to meet the needs of different customers, certain software components need to be removed while other components are added to extend the functionality of the system. The ability to quickly add these software components into the system, without incurring additional costs, and eliminate unused modules relies heavily on the configurability of the software and hence the RTOS itself.

An RTOS with a configuration method detailed above provides many advantages in embedded system software development. These advantages include:

- Creating faster applications because variables do not have to be checked during runtime to determine what action to take.

- The software is more responsive and latencies are reduced, which aids in creating a more deterministic system, which is very important in real-time devices.
- The code base that is generated is simpler, making verification and testing easier.
- The software is tailored for the application, allowing developers to create an application-specific real-time operating system.
- Costs are reduced since resource usage is optimized and processor cycles are used efficiently, thereby enabling less expensive hardware to be specified in the design.

Porting and Support

Ease of porting the software onto the actual hardware platform used is a concern and is a serious consideration for RTOS evaluation. Properly architected software solutions can simplify the porting process by providing support for a wide variety of popular embedded processors as a starting point to the new hardware port. This hardware support also provides a means to get a head start on the software development while the actual hardware is being built.

The ideal RTOS should abstract the hardware-specific software components away from the higher-level software modules to simplify the porting process, as well. Providing the higher software layers creates a common interface to the hardware and eases application porting, regardless of the hardware implementation.

An example of an RTOS providing a hardware abstraction layer to ease porting is shown in Figure 1. As seen in Figure 1, a software module, shown as the Hardware Abstraction Layer (HAL), is situated directly on top of the hardware. The HAL provides a standard Application Programming Interface (API) for high-level software to call into, shown as the Application Layer in the Figure. This allows the low-level software to change without changing the Application Layer software.

For example, the HAL provides a function `hardware_reset`. Within

the HAL, the `hardware_reset` function is implemented, which gets the processor to restart program execution. The Application Layer calls the `hardware_reset` routine when it wants to reset the hardware. When it comes time to port over to a new hardware platform, the HAL that supports the new hardware implements a different `hardware_reset` routine. However, the name of the routine remains the same, only the functionality within the `hardware_reset` routine changes. Therefore, the application does not need to be modified every time the hardware platform is changed.

With all RTOSes, support is critical. But with open source, getting support is always a key concern when determining whether a certain open source RTOS should be used. Help can be obtained from an open forum, relying on the developer community for answers, or via paid support. The developer community can provide adequate support since most developers encounter the same problems during the development cycle and are eager to share this knowledge. However, it may be necessary at times to elicit the help of a company with expertise in a particular RTOS for a solution to a specific problem and assure timely responses to questions.

One example of using the developer community for technical support might occur when a generic driver for a particular family of flash device, such as the Intel 28Fxxx, is provided by the RTOS. However, the developer requires instead a specific device driver for the 28F320C3. In this case, if the software module is not available for this device in the current RTOS release, the developer can query the developer community to see if others are working on a driver for the particular device needed. If another developer is writing software for the 28F320C3 flash device, the open source work done can be leveraged and incorporated into the design. When community support is unable to provide the needed solution, it is still great to use an open source RTOS solution so the developer can go in and make the necessary modifications to the code herself.

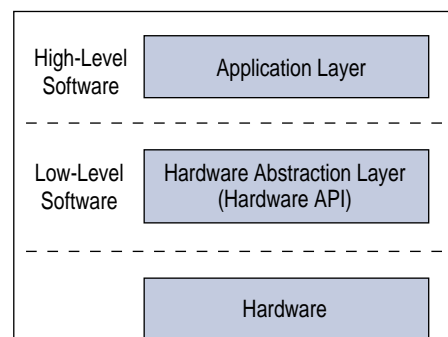


Figure 1

These blocks show how providing a Hardware Abstraction Layer in the software can ease the process of porting an application software to new hardware platforms as well as adding different hardware components to a system.

Costs and Licensing

One issue that is not specific to embedded systems, but to development in general, is cost. It is crucial to keep any up-front fees down while eliminating any royalties associated with the software used in the system.

Many RTOS solutions today have up-front fees based on the number of developer seats needed. These fees can quickly enter into the tens of thousands of dollars before even one line of code is written. This essentially locks the developer into a particular RTOS even though the software might not be right for all embedded development projects. Furthermore, these fees are due each year that the software is used thus creating quite a burden on any budget.

Royalties for software can be extremely costly, especially for high-volume products where every nickel counts. When cost reducing measures are taken, removing a royalty is not an easy feat. In some cases, per unit royalties are tacked on for software modules that are used to extend the functionality of the base RTOS solution. For example, some RTOSes do not provide a networking stack as a standard feature with the core software. Instead it is an add-on component, often by a third-party vendor. In these cases, the third-party supplying the networking stack charges a royalty for the use of their software as well.

The Softer Side

Care also needs to be taken when evaluating the licensing requirements for a given RTOS. Some licensing terms can force proprietary portions of application code to be released to the open source community. Some licenses are “infectious” in a way, even causing software modules linked with a particular piece of open source software to be released for others to use. In a typical embedded pro-

ject, which is very different than a desktop software development environment, the lines can become blurred between the different software modules in a system. The RTOS license should not force particular proprietary aspects of an application to be opened up for public use. This can hinder a company’s ability to maintain a competitive edge.

Possible Solutions

There is a wide range of RTOS solutions available, both proprietary and open source. Weighing the pros and cons of each possible product against the less-than-obvious selection criteria discussed above aids in the decision process. Of course, the specific requirements of the embedded application must also be factored in.

Embedded Linux

The Linux kernel was designed as a desktop operating system and not initially intended for embedded real-time systems. Because of this the embedded Linux kernel may not provide the responsiveness needed to meet the real-time constraints of certain embedded systems, and the device drivers included with Linux are geared toward a PC environment.

Many embedded Linux distributions offer the ability to configure certain components of the kernel and other software modules. However, the embedded Linux kernel resource requirements might not be scaled down enough to fit into smaller embedded devices. Some distributions demand as much as 8 Mbytes of RAM and 4 Mbytes of ROM. This might not be an issue for higher-end systems such as routers and set-top boxes, which provide ample memory and processing power; however, these still come at an increased cost.

Depending on the distribution, the configuration system offered with embedded Linux might not provide the detailed level of configurability needed for an embedded system. For example, a distribution could require all networking modules to remain in the system if an Ethernet device driver is incorporated into the software. However, the particular embedded application might not want the networking software components included with the Ethernet driver since a proprietary protocol is using the Ethernet interface rather than a standard networking stack. In this case additional software is included in the system that is not used and not wanted, causing debugging and testing concerns.

Because of growing interest in Linux, a large developer community exists. This

is a big benefit when looking to staff for a project or seeking support from the developer community. The open source nature of Linux is great for getting support for technical problems from other developers that have encountered similar issues.

One of the first steps in using embedded Linux is selecting a distribution that meets the project requirements. Typically, a distribution is used to leverage the work done by a company in porting Linux to particular platforms and providing development tools for configuring the Linux kernel and its toolset. There are costs associated with using a distribution since this support is not provided for free. This could also have the same ramifications as other commercial RTOSes on the market, where the developer is now locked into a particular solution once a specific embedded Linux distribution has been chosen.

Commercial RTOSes

Most commercial RTOSes do not offer the ability to configure the software adequately for embedded software development. This leads to a waste of resources, which is very costly in embedded systems. For example, a commercial RTOS solution may offer a flash file system that includes a device driver for reading and writing to a flash device. This entire software component is typically incorporated into a single object module designed to operate as a solitary unit. But if your design required only the flash device driver, but not the higher layer file system, you would need to go into the code and hack out the unneeded software functionality. With a configuration system, the developer is able to select the individual software modules that are included in the project's build image and layer them to meet the application's specific requirements.

Another concern when using a commercial RTOS is that a single source rather than a developer community provides support for the software. This means the developer is at the mercy of this company for when they can answer problem queries—often smaller companies suffer inadequate technical response

What is eCos?

eCos, which stands for the Embedded Configurable Operating System, is an open source and royalty-free real-time operating system. The eCos home page can be found online at: <http://sources.redhat.com/ecos>.

A complete development and debug environment is included in the eCos system, based on the GNU tools (compilers, assemblers, linkers, debuggers and simulators). Configuration and build tools are available to run on either a Linux or Windows host environment.

eCos is ideal for use in real-time applications. The eCos kernel provides standard RTOS features such as pre-emptable threads with multiple priority levels, low latency interrupt handling, and multiple synchronization methods. The eCos core functionality also includes interrupt and exception handling, memory management and a wide variety of device drivers that include devices typically found in embedded systems. These drivers include serial, Ethernet, flash ROM, USB and PCMCIA devices. There is an I/O Application Programming Interface (API) included to perform basic functions such as sending/receiving data and configuration.

times due to the fact that other larger companies are monopolizing the technical support staff. The developer is also relying on the knowledge of the support staff, rather than the other developers or users of the software. When using a developer community, project engineers and programmers are able to get answers from developers that have actually implemented the software and gone through the same process that the programmer is going through. This is invaluable to quickly get correct answers on technical issues.

Since many commercial solutions rely on third-party components to extend the core functionality of the RTOS, the developer is left trying to obtain support from two different sources. If a networking stack is provided by a third-party, any problems with the stack are referred to another company. This often leads to each company blaming the other for the lack of operation of the software.

eCos supports a wide range of 16-, 32- and 64-bit processor architectures, which include :

ARM	Fujitsu FR-V
Hitachi H8/300	Intel x86
Matsushita AM3x	MIPS
NEC V8xx	PowerPC
Samsung CalmRISC16/32	SPARC
SPARClike	SuperH

Along with the core components described above, eCos provides additional functionality to extend the capabilities of any embedded system. The extended functionality includes networking support (including TCP/IP stack and SNMP support), a POSIX 1003.1 compatibility layer, the RedBoot ROM monitor, RAM, ROM and JFFS2 file systems, and PCI and USB libraries.

The open source nature and configurability of eCos allow many third-party developers to enhance the feature set and functionality by providing add-on packages. Other third-party contributions include GoAhead Web Server, Microwindows Port, Embedded SOAP (eSOAP), Kaffe Java Virtual Machine, Bluetooth Stack and WAP Stack.

Finally, many commercial RTOS solutions are not cost feasible. There are often high up-front fees to get started; this makes evaluating a particular solution very difficult and often times impossible for budget limited organizations. Along with this are fees for software components not included in the base RTOS. For example, most commercial RTOS software includes the standard components, such as the scheduler. However, if additional software components such as a networking stack or embedded web server are needed for an application, additional fees are tacked on for including these modules. In addition to that, many times additional royalties are also added for these components, drastically raising the project's bill of materials.

Roll Your Own

This is always a consideration and can sometimes be the only solution when using such a specific piece of hard-

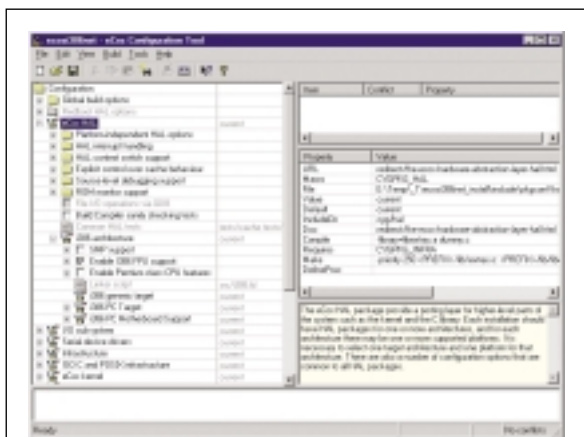


Figure 2

The Configuration Tool eases software component setup to build an application-specific real-time operating system.

however, the amount of time spent debugging the RTOS itself, rather than the application, can quickly add up.

When developing an RTOS, a developer is left with the burden of supporting the software that was created. There are no third-party sources or developer communities to rely on to get technical support. The developer alone is also responsible for developing all support software for the RTOS. And typically, there will not be any configuration method implemented. Each time functionality is added or removed a big effort is needed to determine how the new compo-

nent should be added or how to hack out the unnecessary modules.

eCos RTOS

eCos provides all of the conventional metrics of a typical RTOS but also exceeds other solutions by incorporating additional functionality to support the other non-traditional metrics we've been describing. eCos offers many of the same qualities (open source, royalty-free and so on) as embedded Linux, but is optimal for embedded systems (see Sidebar: What is eCos?).

eCos was designed for portability by using a layered approach for the different software system components. These software components are controlled by an innovative configuration method: eCos uses a source level configuration method to control what functionality is included in the final RTOS image. This provides the developer with control of the software module at the earliest stage, meaning the component itself is built for the specific application that it is intended. Source level configuration offers the best control over resource usage, which is ideal in an embedded system.

The eCos configuration method is also ideal for porting the software over to new hardware platforms. Its modular nature makes adding and removing specific software components easy. If, for example, a new Ethernet port is added to

an existing project, the developer can quickly add in the necessary device driver support for utilizing the new hardware interface.

Support for eCos can be obtained for free from the developer community. The eCos discussion mailing list is provided for technical support and maintained by the original software architects that implemented the RTOS. If more specialized support is necessary, numerous companies are available to provide fee-based individual technical support.

Since eCos is open source and royalty-free, the startup costs for evaluating and using eCos are zero. This is attractive for both small companies that do not have the funds to support high priced software tools and large companies that want to test-drive the software prior to using it.

A Closer Look at eCos

eCos was designed for deeply embedded systems where memory, processing power and other system resources, along with real-time execution constraints, are top design issues. As mentioned previously, eCos uses a unique source level configuration method for constructing the RTOS image.

Let's take a look at an example of the eCos source level configuration method. The following code snippet uses a flag `INCLUDE_FUNCTIONALITY` that is used to either enable or disable a section of software that provides a specific operation:

```
#ifndef INCLUDE_FUNCTIONALITY
/* Software to do something... */
#endif
```

Using this method when `INCLUDE_FUNCTIONALITY` is not defined, the associated code is completely removed from the system. This eliminates the unnecessary code from the RTOS image, saving code space and removing the need to test software that is not used. Source level configuration is made easier by using an application, the Configuration Tool, to control what functionality and features are

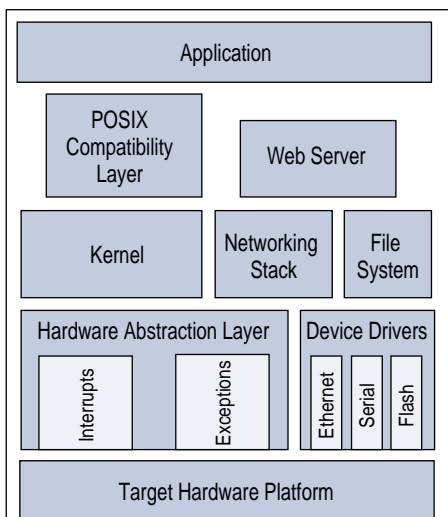


Figure 3

eCos system block diagram includes software components to meet the needs of most embedded systems.

ware that only a customized software solution will do. However, this is a rare case. By developing an RTOS from the ground up, a developer is forced to spend time debugging and testing the RTOS rather than focusing on developing application software, which is actually the end goal. Typically for most embedded systems, it is more productive to leverage off software that is already implemented and tested. At first it might seem that up-front fees are eliminated;

included in the application image at runtime. Figure 2 shows the main screen of the configuration tool.

Using the Configuration Tool, developers can add entire software components, such as a networking stack, and select how specific functionality operates within the software component, such as whether DHCP is used to obtain an IP address. The modularity of the software components within the eCos system makes porting to new hardware platforms easier. Figure 3 shows a system block diagram of different software components in an embedded system.

Since each software module is self-contained, it can easily be removed from the system when it is not needed. For example, if the code needs to switch from a big endian to a little endian processor, the old HAL is removed and a new HAL is incorporated into the system. The higher-level software layers still remain and utilize the new hardware. The book *Embedded Software Development with eCos* includes detailed instructions on porting eCos to new hardware platforms.

For technical support there are six different mailing lists available for the eCos project that provide a forum for various topics of discussion. The mailing lists are responsive and maintained by the original eCos software architects. A number of companies are also specializing in providing individual fee-based support directly specifically at eCos. These companies offer in-depth knowledge of eCos and have experience using eCos across a wide range of target hardware.

The eCos license contains a special exception to the standard GNU Public License (GPL) Version 2, making it ideal for embedded systems. Developers have full access to the entire software source code, including the tools, which can be modified as necessary. This gives the developer tremendous control over her destiny. If an answer cannot be obtained from the developer community, the user can go into the software and seek out the answer by themselves. eCos provides everything needed to set up a complete embedded software development environment for free.

After digging below the obvious criteria for RTOS evaluation, eCos is the best

open source choice for many embedded systems projects. The book *Embedded Software Development with eCos* includes step-by-step instructions for setting up eCos and getting it running. This allows developers to rapidly evaluate eCos for a particular project. Managers are also able to use this book to get an understanding

of its features to meet the requirements in their projects. ■■

*This article is adapted from the in-print book **Embedded Software Development with eCos** written by Anthony Massa, (Prentice Hall PTR, ISBN 0130354732).*