

SELECTING AN EMBEDDED RTOS



FEATURED INTERVIEW:

EXCERPTED FROM WWW.EG3.COM

Prepared by:

eg3.com

Jason McDonald, Senior Editor

eg3.com

tel : 510.713.2150

email : info@eg3.com

web : <http://www.eg3.com>



EXPRESS LOGIC: THE BUSINESS SIDE OF SELECTING AN EMBEDDED RTOS

5 December 2008: The Business Side of Selecting an Embedded RTOS

INTERVIEWEE. JOHN CARBONE
 VP, MARKETING
 TEL. (858) 613-6640
 EMAIL. jcarbone@expresslogic.com
 COMPANY. EXPRESS LOGIC
 WEB. <http://www.rtos.com/>

Q. First of all, John, thank you for agreeing to share your insights with eg3.com's new "Editors' Choice" podcast listeners. Before we dive into the "business side" of selecting an RTOS, can you share with us your background and a little history on Express Logic?

A. I'm a former software developer, having worked on real-time systems back as far as the early 1970's in the AWACS program, and later on telephone circuit switches and nuclear power plant monitoring systems. I got into the field of array processors and scientific computing, and became sales manager and later, marketing manager for board manufacturers and a software tools and RTOS company. My background gives me a developer perspective, which I bring to Express Logic where I contribute to the product planning and marketing, present papers, write articles and generally stick my foot into all sorts of interesting topics.

Currently, I manage the marketing activities at Express Logic. That includes partner relations, marketing communications, and product marketing.

Q. We have another interview in the Guide that focuses more specifically on products, but give us a quick overview of Express Logic, its products and the "business proposition" that it represents for embedded systems developers.

A. Express Logic offers the following products for embedded development. These products are used to develop, and are found in, over 500 million electronic devices:

1. *ThreadX® RTOS*
2. *FileX® file system*
3. *NetX™ TCP/IP stack*
4. *USBX™ USB Host/Device stack*
5. *PEGX™ Graphics toolkit*
6. *TraceX™ Graphical event trace tool*
7. *BenchX™ Eclipse-based IDE*

We offer an easy-to-use RTOS that is field-proven in over 500 million devices. Developer studies have shown that projects using *ThreadX* get completed on or ahead of schedule more often than projects using other RTOSes. Developers find our API to be intuitive and easy to learn and use, which explains why they continue to use *ThreadX* for product after

product. In addition to its outstanding ease-of-use, *ThreadX* offers the following key features and benefits:

1. Full source code - for easier understanding of RTOS operation, and ability to fine-tune for specific application needs;
2. Royalty-free licensing model – for low cost manufacturing;
3. Small memory footprint – for use in limited memory situations;
4. High performance – for fast context switching, low overhead;
5. Broad microprocessor support – over 30 microprocessor architectures are supported, including all the industry leaders in the 32-bit field, and some 16-bit architectures as well;
6. Freedom of choice in tools – including *GNU*, *ARM RealView*, *Freescal*
CodeWarrior, *MIPS SDE Navigator*, *IAR Embedded Workbench*, *Wind River Workbench*, *Green Hills MULTI*, *Lauterbach TRACE32*, and many more.

Q. OK, now on to the question, “What goes into choosing an RTOS?”

- A. Technical criteria for selecting an RTOS are certainly important, but perhaps not as all-important as some might assume - really only 30% of the issue, and most RTOSes satisfy about 75% of embedded applications . For example, there is much emphasis on real-time performance in an RTOS. But, do developers really know just how much performance their application actually needs? If they do, then an RTOS that delivers that performance might be suitable, but is one that delivers more performance actually any better? Many RTOSes might meet the performance needs of an application, so how would one choose among them? Likewise, memory footprint is important. But, if an application (including the RTOS) must fit in 256KB of memory, is it really relevant whether the RTOS occupies 2KB rather than 4KB of that?

So, in my opinion, many times the selection of an RTOS cannot be resolved through technical criteria alone. You may find that several choices appear to meet all your criteria, or at least offer comparable overall technical scores. In those cases, you can consider other factors to find the RTOS that represents the best choice for your project.

Moreover, if the RTOSes you’re considering all fall within a relatively small envelope of technical criteria, a major distinguishing factor outside that realm might overwhelm the small technical advantage one RTOS has over another.

In short –many RTOSes can “do the job” from a technical perspective, but selecting one that you’ll be happy with might very well depend on several non-technical criteria. Now, just what are those non-technical criteria? I think the following are all potentially deal-making or deal-breaking issues that demand consideration:

1. Cost
2. Licensing Terms
3. Legal Risk
4. Time-to-Market

Q. Engineers and technical folk are historically in the driver’s seat for the choice of embedded software. What are the big factors that you think impact the “business choice” of an RTOS? Where are the costs?

A. The big factors are:

1. Cost - This is tri-fold: Acquisition (licensing model), Development (time to market), Deployment (royalties), and Support (latent defects, recalls, routine support)
2. Licensing Terms
3. Legal Risk - Indemnification, GPL sharing requirements
4. Time-to-Market

In addition to the cost factors discussed above, *Linux* carries an additional cost in the legal area. This cost manifests itself as risk related to infringement of the intellectual property rights (IPR) of others. Contributors to *Linux* might (and "might" is the key word here) assert patent, copyright, or other IPR on work they've contributed to *Linux*. Or, they might be accused by others of copying the code that they contributed. This "might" occur sometime down the road after many units have been deployed. What is the cost of a negotiated settlement with such a claim? Even if baseless, lawyers cost money (at rates as high as \$700/hr), and claims can consume many hours. Even if no claims are encountered, a lawyer might need to review licenses and contracts related to your development, to make sure no such claims are likely to succeed, and that itself costs money. Then, there's always the potential that you've missed something.

With a commercial RTOS, generally there is no such risk. Commercial RTOS vendors universally indemnify licensees against any such claims, making all such problems the responsibility of the vendor, and not the licensee. This can be a huge difference between a commercial RTOS and *Linux* or an In-house solution.

Q. What sorts of "gotchas" can you warn developers about as they go about evaluating the big choice categories - build-your-own RTOS, using a commercial RTOS, and using a "free" RTOS like Linux. Help us out with some of the "hidden" costs that people might not see in the RTOS selection.

- A. It's easy to hide costs with creative licensing plans, so developers are advised to calculate "total cost" as the sum of acquisition cost (up-front), Development Cost, Deployment Cost (royalty per unit manufactured), and Support Cost. That eliminates the games that might be played by moving cost from one place to another.

Management might fall in love with a high-level cost benefit (e.g.: *Linux* is "free") and let that blind them to other cost factors that might obliterate that benefit.

Regarding true cost, I suggest consideration of the following components:

Cost Component	Linux	In-House	Commercial
Acquisition Cost – how much does it cost (up-front) to procure (license) the RTOS?	Indeed, in this area, <i>Linux</i> is free. But, only if you download it yourself from the web.	By definition, an in-house RTOS isn't "procured," it's developed in-house. The person-hours required to	Typically, commercial RTOSes are licensed for an up-front fee, with or without royalties (which would show up under "Manufacturing Cost" below).

		design, develop, and maintain the RTOS must be cost-estimated.	
Development Cost – how much will it cost to develop your application using this RTOS?	<i>Linux</i> might score well here, based on its broad ecosystem of compatible middleware and freeware, minimizing what has to be developed in-house as part of the application. Conversely, <i>Linux</i> might suffer here due to its complexity, which will slow down the learning curve and increase the error rate. In addition, if <i>Linux</i> is procured directly from the web, someone must find it, download the right elements, configure it for the target system, and build it. That person's time is a cost.	Depending on how much is put into the in-house RTOS, this component can be less expensive or more expensive as a result. Regardless of the feature-set designed into the RTOS, the availability of compatible middleware from 3 rd parties will likely be non-existent, meaning that all code must be developed in-house. Also, development and debugging tools might not be geared to support the in-house RTOS, making development less convenient.	Commercial RTOSes generally offer an ecosystem of compatible middleware that can be procured (adding to the cost above) for use in your application (decreasing the cost in this area). Development tools generally support commercial RTOSes through kernel-aware debugging, event trace and analysis, and builder integration, all serving to help speed the development process.
Manufacturing Cost – how much does the RTOS cost “per unit” manufactured?	<i>Linux</i> does not require royalty fees per unit, so one might conclude that manufacturing cost is \$0. But, there is a “royalty” for	Typically, the in-house RTOS scores well in this area. No royalties are required (unless your corporation	For commercial RTOSes, this reduces to a royalty vs. royalty-free issue. A royalty-free RTOS may come at a higher procurement cost upfront, but many royalty-free RTOSes are also LESS expensive than royalty-bearing alternatives! The

	<i>Linux</i> in the form of the 2MB of memory that must be added to every product just to hold <i>Linux</i> . That 2MB of memory isn't free. Either it's an extra 2MB, or it's 2MB of existing memory that the application can't use.	charges an in-house transfer fee), and memory size can be controlled to avoid significant expense.	developer is well advised to add both factors to determine true (acquisition+royalty)/volume unit cost.
Support Cost – How much does it cost to support the RTOS?	<i>If Linux was downloaded directly from the web, then all support is your responsibility. That means keeping up with bug fixes, new code, enhancements, and training. These person-hours also represent cost that must be considered.</i>	In-house RTOSes require support from in-house resources, which have a cost. With nowhere else to go for support, all costs for bug fixes, new code, enhancements, and training become yours.	Commercial vendors generally offer full support for an annual fee. Optionally, developers might opt to self-support, especially if full source code is licensed from the vendor.

Another area for scrutiny is tools cost. Some RTOS vendors also bundle tools with their RTOS, even *requiring* the use of their own tools. Such tools have a cost, and that cost might be used to hide some of the cost of the RTOS, making the RTOS appear to be less expensive. For example, which RTOS is less expensive:

	RTOS-A	RTOS-B
Up-front	\$9,000	\$10,000
Royalty	\$0	\$0
Tools	\$6,000/seat	\$1,000/seat

As you can see, such bundles can be misleading if one considers just the RTOS cost. The point is that some RTOSes have an influence on related costs, like tools.

Q. Open Source / embedded Linux was quite the rage a few years ago - the Open Source “free software” story seemed ready to take embedded systems by storm. There were all sorts of arguments about whether Linux “was” or “was not” a true RTOS. Let’s leave that side. What do you see as the BUSINESS perils in going with Linux?

A. First of all, I think Linux, and open source in general, is a great phenomenon that has benefited the community in many ways. Initially, many developers seemed to think Linux could be used in any embedded application, for free. As time went on, over the last few years, it’s become clear that Linux isn’t for all applications, and it isn’t free. Leaving the applicability to applications as a technical issue for another day, let’s look at the non-technical issues surrounding Linux in embedded development.

The first is support. Linux is a large amount of code, and it’s changing daily. Someone has to be responsible for finding the right release, all the relevant patches and updates, and configuring it for the customer’s actual target platform and application needs. That might be considered “technical” in a way, but the impact is financial since that support costs significant money, whether it’s performed in-house or through a commercial Linux supplier.

Second, the legal aspects of Linux are far from clear. There is the GPL issue of exposing application code that’s integrated with Linux, but the far more serious concern is the risk of violating the rights of a contributor, or the owner of the code from which a contributor copied code. Both copyright and patent law protects these rights, and there is no way of knowing who actually owns the code that ends up in Linux. What’s more, with this uncertainty, users are exposed to claims of infringement long after they have used the code and deployed it in their products. Imagine the expense of recalling those products to remove the offending code? Commercial RTOS companies, like Express Logic, indemnify developers against any such risks. 100%. Nothing to worry about. But, as far as I’m aware, that indemnification is not provided by the OSF, or by vendors of commercial Linux.

Q. Many of the Linux vendors like MontaVista or Wind River would claim that they have “solved” the business problems with Linux by blending it with a commercial market. Play devil’s advocate for us here - what sorts of issues would you point out for a business manager that was considering Linux, and considering a “commercial Linux.”

A. There’s a huge difference between Linux and Commercial Linux. First of all, Linux is completely exposed to all the legal, integration, and support issues we discussed above. Commercial Linux addresses the integration and support, but at a price. What attracts many to Linux in the first place, its “free” acquisition cost, no longer is there. On top of that, do MontaVista and Wind River indemnify their customers against copyright and patent infringement, trade secrets, and the like, that might be present in the open source code? Commercial RTOS vendors like Express Logic do, so there is no risk. But without such indemnification, commercial Linux customers pay, possibly more, and still have the legal risks to manage themselves.

Patent and Copyright infringement should be concerns of every software developer. It’s generally considered unethical to copy someone else’s work and pass it off as one’s own. The ACM states exactly that in its code of ethics (<http://www.acm.org/about/code-of-ethics>). Sadly though, it’s not always illegal. If you decide to copy any part of someone

else's work, you should make sure to seek good legal advice regarding any potential liability. Conversely, if you use a product that includes work copied from another, you may be liable. Again, you should seek legal advice to assess the risk. As mentioned above, commercial indemnification is one way to bound these risks, or eliminate them altogether. Without such indemnification, how can you be certain that the software you're using doesn't infringe someone else's rights? In effect, you're indemnifying yourself!

Q. What about the “do it yourself” model? There are still, today, a large percentage of our own survey respondents who are “building their own RTOS.” In what situations do you see this as a good choice? In what situations is this potentially a disaster???

A. I think it may be a good choice if the needs are so minimal that an “executive” or “kernel” can be developed in less than a man-month or so, which would mean only about 500 lines of code. I'd estimate that a man-month would cost most companies, fully burdened, about \$10,000 - \$20,000. Still, this is likely tied to an individual architecture and not easily adapted to other architectures. But, generally, if the requirements of the system are extremely simple, it's not necessary to use a commercial RTOS.

Now a recipe for disaster might be a bit of underestimation of the scope of work, a change in processor selection, a hand-off from individual to individual as enhancements are added, changes in requirements as the product nears completion, and of course, the piece de resistance – the developer responsible for the RTOS quits at the worst possible time. Almost any one of those could lead to disaster.

Q. Let's assume a design team has decided to “go commercial.” What factors would you suggest they compare and contrast their various commercial choices on? From a business perspective, where are some of the points-of-differences among commercial RTOS companies like Express Logic, Microsoft Embedded, QNX, Quadros and others?

A. In general, we might group commercial RTOSes into 3 major categories: Miniature desktops, Military/Large Systems, and Small - Consumer/Medical/Industrial. The miniature desktop group includes Microsoft, the military, telecommunications and large system vendors include Wind River, Enea, QNX and GHS, and the small group is where Express Logic operates. These groups exhibit similar characteristics:

- a) Mini-desktop – Windows for PDAs, Smartphones, etc.
- b) Large – Virtual memory, large memory footprint, robust service repertoire
- Small – Linear address space, small memory footprint, simple set of services

It's important for developers to examine their own needs and select from the group that best matches those needs. It's not wise to select a Large RTOS for a simple application, because that just introduces unnecessary complexity that slows learning, reduces ease of use, and adds overhead and size.

The criteria mentioned above:

1. Cost - This is tri-fold: Acquisition (licensing model), Development (time to market), Deployment (royalties), and Support (latent defects, recalls, routine support)

2. Licensing Terms
3. Legal Risk - Indemnification, GPL sharing requirements
4. Time-to-Market

Represent the key business issues that must be examined and evaluated. In addition, vendor stability, reliability, reputation and pedigree is a huge consideration. Developers will end up with an in-house RTOS if their commercial vendor disappears or decides to stop supporting the RTOS for some reason. As a partial protection against this, licensing full source code can be very comforting, especially if the RTOS is relatively simple. Still, the company stability should be considered in the following areas:

a) How long has the company been in business	If <5 years, a more careful analysis might be warranted
b) Is the company profitable	Unprofitable companies are going out of business – it's just a question of when. Public companies must publish P/L reports quarterly, while private companies do not. Ask the vendor for this information, especially if in business less than 5 years.
c) How widely has the RTOS been used	The more deployments the better, as high volume translates to field-testing and validation. A widely deployed RTOS is less likely to have latent bugs, and is more likely to facilitate successful development and timely product completion.
d) What do other users say about the company and the RTOS	Check forums, message boards, and the company's own web site for customer testimonials and comments.

Q. Engineers often want access to RTOS source code. Do you think that there are business reasons why this might be important? How so?

A. Apart from any technical reasons, source code provides a “safety net” in the event of vendor abandonment. In the “olden days,” companies would pay for “bonded storage” of source code, to be freed up upon bankruptcy or other catastrophic event. Today, source code is generally available at an affordable cost. This makes sense as a business move, but also has technical benefits for developers.

Q. What about the hardware and software “ecosystem” around an RTOS? How important do you think partnerships are in terms of the developer perspective? How is he to assess whether Express Logic has a “strong” or “weak” partner ecosystem?

- A. The ecosystem tends to be more important as the complexity of the application increases. Some developers can license everything they need from the vendor, and develop the rest themselves. Others prefer to integrate 3rd party components and focus their value added somewhere else. The right balance will vary greatly from company to company, and depending on the product being developed.

Assuming the ecosystem is important, developers can investigate it easily. Most RTOS vendors expose their ecosystem partners on their web site. Others will make such information available on request. Still others can offer “just-in-time-integration” to meet customer needs. In any case, this is worth exploring as part of any RTOS evaluation, but tempered with the realities of your actual project needs. Remember, making a decision based on speculative needs might result in the best solution for the problem you never face. Better to make sure you solve the problem you know you have, then to worry too much about what you MIGHT need down the road. Of course, the right balance between the two is best.

- Q. Thank you for this interview.**