Embedded Systems: A Primer

by Geoff Bessin
Senior Technical Marketing Engineer
Automated Testing
Rational Software

In the world of software development, the industry can be divided into three large sectors: embedded systems/network-connected devices, infrastructure, and e-business systems.

Would it surprise you to know that both of the first two sectors (infrastructures and embedded systems/network-connected devices) rely heavily on embedded system technology? It's true. But how important are they, really? Well, if you've ever called someone on your cell phone, you have an embedded system to thank. If you've ever driven a car or landed safely in an airplane, thank embedded systems. The same goes for using your Personal Digital Assistant (PDA), benefiting from complex medical equipment, or consuming electricity from your local power company. Embedded systems are everywhere.

Impressive? So are the numbers. Every year, more than a billion dollars are spent on the technology and know-how necessary to make all of these systems a reality.

Embedded systems have a vocabulary and grammar all their own. If you are new to embedded systems, this primer will give you useful background and a basic understanding of the underlying concepts.

What Is an Embedded System?

Tough question, really. There is no one answer. I asked at least a half dozen industry experts and got as many answers. In fact, it was so hard...
to pin down a definition that I almost started thinking "embedded system" was just another term for "software." Nevertheless, there's one fact about embedded systems that all the experts do seem to agree on:

An **embedded system** is any software system that must be designed on a platform different from the platform on which the system is intended to be deployed.

What is meant by *platform*? On the development side, *platform* typically refers to an operating system capable of hosting software development, such as Windows, Solaris, HP, etc. On the target side, the word *platform* refers to the devices on which the embedded system will be deployed.

OK then, why the design constraint? Why aren't embedded targets capable of hosting software development? Because these targets are optimized for performance and/or simplicity, they lack the equipment necessary for development (such as keyboards, monitors, networking, etc.). In general, development for the embedded environment is referred to as "cross-platform development."

**Embedded System Performance**

To reiterate, target platforms, the actual embedded devices and systems themselves, are optimized for performance and/or simplicity. What do we mean by performance? If you think it means the target has to be extremely fast, then you're only partially correct. For most embedded systems, a maximally acceptable process time is rigidly observed, but that time need not be super fast -- just fast enough for the system in question. Consider the airplane: How much time is acceptable between a change in external atmospheric conditions and the appropriate alteration made by the autopilot? Perhaps the reaction need not be lightning quick, but it certainly can't be sluggish.

And how about simplicity? Depending on the responsibility of the embedded system, small, inexpensive, low-power microchips are viable and preferable options to more bulky alternatives. Did you know that there is more computational power in a modern-day automobile than there was in the Apollo 13 moon lander? Can you imagine the cost and size of today's automobiles if their embedded systems were complex and bulky?

So what's the overall responsibility of an embedded system? Again, I found the experts could agree on this statement:

The **objective** of an embedded system is to execute as fast as necessary in an asynchronous world using the smallest amount of code and with the highest level of predictability. (Note: Predictability is the embedded world's term for reliability.)

How could this even be possible? By definition, in an asynchronous world, inputs are unpredictable; in such a world there is no such thing as a reliable, dependable, predictable environment!

To understand, think in terms of your car: Every time a piston in the
engine expands, your car has to measure temperature and pressure to decide how much gasoline and air to inject into the cylinder before the next compression. If you drive your car at 130 mph on a deserted highway, it will force your car's embedded computer to make that decision thousands of times per second because your car will be running at 6000 RPM. Every reading will supply different values that require different decisions. But your embedded computer will let you enjoy that 130 mph ride because the system itself is very predictable, very reliable. It will even stop you when the engine is too hot or the car is out of gas. And it had better be predictable and reliable -- you wouldn't want to reboot your car while breaking the sound barrier, would you?

Okay, let's review. We now understand that embedded systems:

- Must be developed in a cross-development environment;
- Implement functions that provide an element of control in an asynchronous world;
- Use the smallest amount of code to achieve the greatest necessary speed and degree of predictability possible.

**Target Platforms**

For the simplest target platforms -- platforms that are inexpensive and consume minimal power -- all functions are considered equally important, so the system has no recognition of priority. Functions are typically written in C and then "burned" into some Read Only Memory on an 8-bit or 16-bit microprocessor. There is no operating system and perhaps 0.5K of Random Access Memory for some minor use. It's information in, information out. Lightning quick. (The bit size just refers to the amount of memory accessed by the processor. All memory is divided into little storage bins with individual addresses. The more bits supported by a processor, the more addresses that can be specified.)

As the need for features increases and/or as the need to establish priorities arises, it becomes more important to have some sort of decision-making mechanism be part of the embedded system. The most advanced systems actually have a tiny, streamlined operating system running the show, executing on a 32-bit or 64-bit processor. This is called a Real-Time Operating System (RTOS).

An RTOS runs your PDA. More than one thing may be happening at the same time on this device, and it is the job of the RTOS to direct traffic. But why *real-time*? Here's what my experts had to say:

In the embedded industry, *"real-time"* is used to refer to a system that must compute a result, based on certain inputs, in less than a maximally acceptable time -- i.e., it must respond in real time.

The fact is, real-time response is not always necessary in an embedded system. Nevertheless, within the industry, the term *real-time* has become almost synonymous with *embedded system.*
Embedded System Certification

Embedded systems associated with safety and mission-critical applications typically have extremely rigorous certification standards. For example, all international aviation administrations disallow the use of a new embedded system within airplanes until that system has been developed and tested with the techniques and methods defined within a document labeled DO-178B. Critical to this well-defined process is an insistence on exposing and testing code, based upon code coverage up to the level of completeness software developers refer to as MC-DC (Modified Conditions - Decision Conditions).

So how do you test and certify such critical embedded systems? Your testing toolset must:

- Execute its tests directly on the target system.
- Support all potential development platforms and development tools.
- Accommodate, and not interfere with, the potentially limited memory and process power of the target system.
- Enable everything from unit testing (i.e., testing of functions, classes, tasks) to overall system testing.
- Provide code coverage metrics up to the rigorous MC-DC standard.

With the addition of Rational Test RealTime, Rational can now supply a toolset that meets all the requirements on this list. Of course, not all embedded systems must meet such rigorous certification standards, but testing tools that can meet a strict certification standard can easily accommodate simpler, less restricted, embedded systems.

That's it! Now that you know the basics about embedded systems, you can understand what the big fuss is about. Future Rational Edge articles will provide information about how to apply the tools Rational supplies to develop and test embedded systems, so stay tuned!

For more information on the products or services discussed in this article, please click here and follow the instructions provided.

Thank you!