



REAL-TIME SOFTWARE FOR PC/104 SYSTEMS

Rick Lehrbaum
Executive Vice President, Strategic Development
Ampro Computers, Inc.

A survey of embedded system developers using PC/104 is extremely unlikely to indicate that they selected the PC architecture due to its raw computational horsepower or architectural elegance! Instead, you'll no doubt find that one of these pragmatic factors was behind the decision to use PC/104:

- What the PC lacks in hardware sophistication, it more than makes up for in software abundance. This includes operating systems, drivers, function libraries, and development tools. In other words, it isn't hardware, but the *software*, that dictated the choice.
- PC/104 modules are a lot like Lego[®] building blocks; they make it easy and inexpensive to design, prototype, debug, and deploy an embedded system. Thus, the hardware mirrors the object-oriented methods of the software.

Which of these reasons is more important depends on the specific application. However, it's often stated that software, rather than hardware, has increasingly become the dominant cost and risk factor in embedded system development.

This being the case, it's useful to explore the software options available to developers of PC/104-based embedded systems.

EVALUATING YOUR OPTIONS

In theory, embedded applications should follow the "well behaved" software model shown in Figure 1(a). However, the relatively modest performance of both the PC hardware and the MS-DOS operating system have prompted many programmers to circumvent MS-DOS and instead interface directly with the PC's BIOS or device drivers as indicated in Figure 1(b). Furthermore, whenever performance is critical (such as display updates), programmers tend to take direct control of the hardware, as in Figure 1(c).

There are many ways to structure the software side of your PC/104 application. Here are several possibilities:

- "*Standalone*" applications — perhaps you don't need an operating system at all! Did you know that a standard PC-compatible BIOS offers you a way to run your application directly out of an EPROM without requiring DOS, device drivers, or anything else?
- "*Normal*" DOS-based applications — this, of course, is the most obvious and straightforward approach. Write your application as a DOS "EXE" file, and include a bootable DOS drive, with your application program on it, in the target system.

- *“Real-time” applications* — if your application demands multi-tasking or high performance software features, you may want to make use of real-time support software. There are several suitable approaches, depending on your system’s needs.

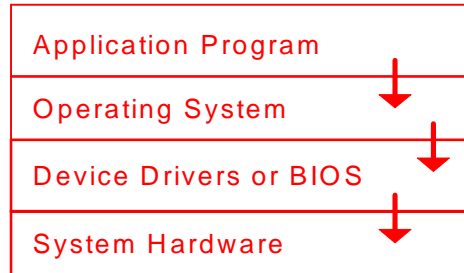


Figure 1(a): “Well behaved” software architecture

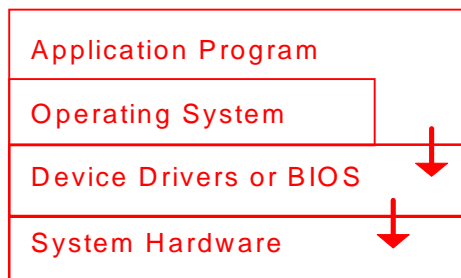
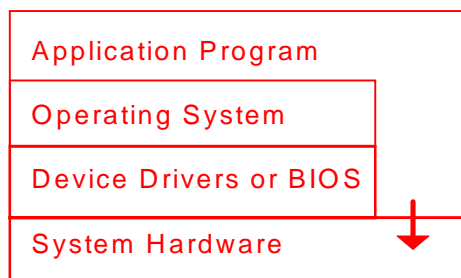


Figure 1(b): Bypassing the operating system



**Figure 1(c):
Bypassing the operating system, BIOS, and device drivers**

KEEPING IT SIMPLE

When you use a single-chip microcontroller you normally burn your code into an EPROM, after assembling or compiling it into separate code and data address spaces. You can deal with your PC/104-based application in a similar manner. The PC architecture includes a handy mechanism that lets you run code from within an EPROM on system startup. This technique, known as a “BIOS extension”, is a standard feature of the PC BIOS. The VGA video BIOS software used to control the display of most PCs is loaded from a BIOS extension.

By having the application software located in an appropriately formatted EPROM, a PC/104-based system can operate out of a BIOS extension. However, don’t forget some basic limitations of this approach. Since BIOS extensions activate prior to booting DOS, the application won’t be able to take advantage of either DOS services or DOS-loaded device drivers (or TSRs). It can, however, use BIOS services. Also, you may not be able to use your favorite compiler, since many compilers need a DOS runtime environment to execute their compiled code.

DOING DOS

The typical PC/104-based application is written in C, compiled into an EXE file, and run under DOS. Plain old DOS. Why use DOS?

Even if the application, once running, has no need for any operating system services whatsoever, DOS may offer some important benefits:

- The presence of DOS as a runtime environment allows you to use your favorite desktop-PC compiler and debugging tools, which also means you can develop and deploy your embedded application without any special embedded tricks.
- Having the same OS in both the “target” and “development” environments substantially simplifies the process of porting the application program from the development system to the embedded system.
- Even if the embedded application doesn’t itself require DOS services, the ability to use DOS or DOS-based utilities in the embedded system can be important. Perhaps, for system installation and maintenance. For example, you could use the MS-DOS INTERLINK utility to transfer data into, or out of, the embedded system.

Frequently, DOS is used for its ability to read or write data to/from a floppy or hard disk drive. For example, an “intelligent” PC/104-based paint mixing machine might read a color matching parameter file from a floppy; or a PC/104-based environmental monitoring instrument might log data that it collects onto a hard drive for subsequent analysis.

GETTING REAL

Complex or mission critical applications may demand operating system features beyond those available from DOS. A DOS-issued system status message like “Press F1 to continue . . .” would be unacceptable in a PC/104-based systems that doesn’t have a screen to display it on, and doesn’t have an F1 key to press!

Turning a PC/104-based system into a “real-time machine” is the sole purpose of a large number of real-time software products that offer a wide variety of excellent alternatives. These real-time software products can be classified three ways, as real-time *libraries*, real-time *executives*, and real-time *operating systems*.

CHECKING OUT A LIBRARY

The simplest way to add real-time support to a PC/104-based application is through the use of a real-time, multitasking function library. There are several benefits to this approach:

- You won't need to deal with the complexities of a more full-featured real-time executive or operating system.
- Function libraries often include full source code, which may prove invaluable in debugging the application or in optimizing its performance.
- The initial expense is likely to be very low, and you probably won't have to pay royalties.

On the other hand, the functions offered by a real-time function library will be limited to features such as task management, task switching, queue management, and event synchronization.

Consider other approaches if you need higher order services such as memory management, a file system, console I/O, serial communications, or networking. In short, if you need basic multitasking support for your real-time application, and are prepared to do all of the hardware management work yourself, this may be a reasonable approach.

MAKE AN EXECUTIVE DECISION

Although they offer significantly more system support than multi-tasking libraries, real-time executives typically remain “lean and mean”. They are thrifty in their use of resources — both hardware and software — and are performance-oriented as well.

Also, most real-time executives are offered in a modular format, allowing you to use (and pay for) only the functions you require. The core component, called a “kernel”, provides basic system and task management and control, including resource initialization, task queuing and switching, memory management, and so forth. Additional optional software components generally support console I/O, serial communications, networking, disk read/write, etc.

Depending on the vendor, source for both the kernel and device drivers may be available — making it practical to troubleshoot sticky problems yourself. Also, some real-time executives are licensed on a project basis, thereby eliminating the overhead and cost of royalties.

Another important feature of most PC/104-oriented real-time executives is their ability to support operation *along with* DOS. Here's how this works: the system boots from DOS, and then loads an EXE file that contains both the executive and the application program. Real-time executives with this capability provide a function call so the application can use DOS or BIOS services (for example, for file or console I/O). A major advantage of this approach is its ability to take advantage of DOS device drivers and TSRs. For example: an application (running under a real-time executive) might access a remote disk drive across an Ethernet LAN; this would be based on DOS-provided file services and DOS-loaded network drivers.

Keep in mind some limitations of DOS and BIOS. Both are non-reentrant, so only a single task is permitted access to BIOS or DOS functions at a time. To prevent problems, the real-time executive must provide a suitable “locking” mechanism. Although this restriction may be unacceptable in some systems, quite a few PC/104 applications don’t need shared access to peripheral resources such as a disk, display, network, or keyboard and can therefore benefit from this approach. Another problem exists if the application requires the use of protected mode, since taking advantage of BIOS or DOS services would require constant switching in and out of real mode — an unacceptable inefficiency.

RTOS as MVP

At the high end of the scale, are the full-featured real-time operating systems (RTOSes).

What’s the difference between a real-time executive and a real-time OS? Basically, an *executive* manages the computers basic resources, including CPU, memory, interrupts, timers, and DMA. An *operating system* does all this, but in addition manages the system’s peripheral devices, which typically include disk drives, keyboard and display, serial communications, and networking.

Many RTOS offerings are modular, consisting of a real-time kernel plus an assortment of optional device managers. Such a product might fit the definition of an executive in its minimal configuration and of a full-fledged RTOS when its peripheral device managers are included. Consequently, the distinction between real-time executives and RTOSes can be blurred.

Whatever it’s called, an RTOS will generally offer the following key features:

- The basic kernel services of a real-time executive (described above).
- Reentrant, multi-threaded replacements for the functions of the normal PC BIOS.
- File system, console I/O, communications, and networking functions.
- The ability to do all this in “protected mode” (on 386 CPUs, and up).

DEVICE DRIVER HEADACHES

No discussion of RTOSes would be complete without a few comments about device drivers.

If your RTOS-based application needs to run in protected mode, all device drivers will need to be written in 32-bit, reentrant code. This means that the RTOS can’t make use of the PC-compatible BIOS, since the BIOS doesn’t meet this condition. So the RTOS will have to substitute its own equivalent functions for what is normally included in the PC BIOS.

This is both a blessing and a curse. You can expect improved speed and robustness. However, any non-standard hardware functions in your system’s PC/104 hardware are going to require protected mode drivers for the particular RTOS and specific hardware you’ve chosen to use. Any PC/104-oriented RTOS is likely to include a complete set of drivers for *standard* PC-compatible controllers and peripherals.

On the other hand, support for embedded-PC extensions such as watchdog timers and solid state disks is less certain. Additionally, popular interfaces such as PCMCIA, SCSI, and Ethernet have

failed to achieve hardware standardization in the desktop PC world, and so may also represent a challenge. It's not uncommon to end up haggling with both the RTOS and hardware suppliers over whose responsibility it is to supply the missing drivers! Although a number of attempts have been made to alleviate this situation by standardizing RTOS drivers, the problem remains an important consideration (see box, "UBDs: answer to RTOS prayers?").

In some cases, you can circumvent the RTOS driver problem by using a real-time executive along with DOS, instead of using an RTOS. However, this only works if your application doesn't need to run in protected mode and doesn't require multi-threaded access to the resources in question. While many PC/104 applications do meet these constraints, others do not.

WHY NOT DO WINDOWS?

Why not use Windows, the world's most popular multi-tasking operating system? Clearly Windows has the ability to manage system memory and protected mode, run multiple tasks, support graphics functions, and simplify user interfaces.

So, why not use Windows (or Windows 95) in a PC/104-based embedded application? In fact, quite a few PC/104 applications, both completed and under development, do use Windows.

What kind of real-time performance does Windows offer? The answer to this really depends on what constitutes "real-time" for the particular application.

To be called "real-time", a system must act deterministically. For some applications, this requires millisecond-level responses; for others, it may mean handling a single event per hour.

Looked at this way, why can't Windows (or Windows 95) support real-time operation in some applications? Perhaps it can. Consider an information kiosk at an airport, whose purpose is to provide information about local entertainment, shopping, and community services. The system might consist of: a 486 PC/104 CPU module, running Windows 95; a touch screen that emulates a standard serial mouse, connected to COM1; a VGA-compatible color LCD display; a SoundBlaster-compatible PC/104 module for speech and sound output; an IDE hard drive, containing the information database; a SCSI CD-ROM drive, for audio/video effects and entertainment; a small thermal impact printer, connected to LPT1, to generate hard copy of selected information or sale coupons; the database is periodically updated from a central office over a modem connected to COM2.

In this example, millisecond response times aren't required, so you can take advantage of the world's most popular multitasking operating system: Windows. If performance is a problem, simply swap out the PC/104 CPU module for a faster one. While this year, PC/104 tops out at a 100 MHz 486DX, don't be surprised to see a PC/104 CPU modules based on Intel's Pentium processor.

If, unlike this example, your application demands quick response to stimuli, efficient use of resources, or "ultra-high" reliability, better use an RTOS instead.

DON'T BE UNBIOSED!

With all this discussion of application and operating system software, don't overlook a very intimate part of every PC/104 system: the CPU module's BIOS. After all, the system depends on the BIOS to perform some important functions:

- Initializing the CPU, memory, and system/peripheral controllers.
- Performing the system power-on-self-test ("POST").
- Loading BIOS extensions, if present.
- Booting the operating system, if present.
- Providing hardware interface support during system operation, for keyboard, disk, serial, parallel, and timer functions.

These are important things to get right! What if, on power-up, a system whose purpose is to perform critical control or monitoring functions doesn't successfully boot its operating software? It might be a security surveillance system. Or, a temperature or pressure monitoring device whose failure could endanger lives or property. Maybe, an intelligent vending machine whose down time costs its owners substantial revenue.

What if a hard disk drive spins up a bit too slowly following a power brownout. When the BIOS fails in its first attempt to boot the operating system from a not-quite-ready disk drive, a normal desktop PC system will likely hang forever — waiting for someone to press an "F1" key, or in some other way reset it. Such a failure mode is undesirable, even unacceptable, in many (most?) embedded applications.

Your PC/104-based application may be required to perform its task around-the-clock, without failure. In these applications, "Press F1 keys to continue" or "CMOS checksum error, Press F2 to enter SETUP" won't do! Fortunately, some PC/104 CPUs now include BIOS enhancements that address many of the special concerns of embedded applications. Options available include battery-free startup, watchdog timer support, fail-safe boot, fast boot, solid state disk (SSD) support, serial console, serial software loader, and system customization extensions.

AND THE WINNER IS . . . ?

Given all these options, what's the breakdown of software architectures for PC/104-based embedded systems? The overwhelming favorite is plain old DOS and Windows. Estimates vary, but it is likely that 75% to 85% of these systems use DOS, with the remaining 15% to 25% divided among half a dozen real time executives and operating systems.