



Real-time OS for multicore processors: Serious implications for AdvancedTCA systems

Intel's multicore processor technology has begun to appear in the latest laptop and multimedia PC systems. The technology will also be making its way onto a wide variety of AdvancedTCA system boards for embedded applications. However, without the right software solution for an embedded environment, how successful will the technology be for AdvancedTCA systems?

TenAsys, Beaverton, Oregon has a real-time OS traditionally suited for executing real-time tasks in conjunction with the Microsoft Windows environment on embedded single and multiprocessor systems, and has been validated on Intel multicore processors. This real-time OS, called INtime, works in conjunction with the Windows OS to control real-time tasks while communicating with the Windows environment for graphics and disk support for the non-real-time parts of embedded applications. In this *Software Corner* column, we will be taking a look at the INtime real-time OS and how it operates on these new multicore processors. A big thank-you is extended to Paul Fischer and Kim Hartman of TenAsys for meeting with me to discuss the technical aspects of the solution as well as Bob Patterson of MKTX for coordination efforts.

TenAsys real-time OS history

TenAsys (pronounced ten-AY-sis) has its roots with Intel and the iRMX real-time OS. The iRMX OS was originally developed and maintained by Intel to support the Multibus market. RadiSys Corporation acquired the Multibus business from Intel, which included iRMX. RadiSys Corporation later spun off the real-time OS products as TenAsys was formed.

In addition to the traditional iRMX Multibus business, a new breed of embedded applications began to develop. These new, loosely coupled applications featured an embedded system connected to a Windows PC as shown in Figure 1.

In this loosely coupled configuration, an embedded system running a real-time OS would report back its information to an application running on a Windows PC where the results would be processed, stored, and reported, and where a graphical operator interface would typically be implemented. The devices were coupled through a serial port, shared memory on the same board, or some kind of bus interface, and eventually, Ethernet.

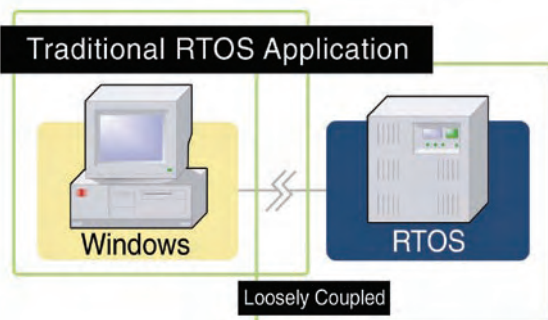


Figure 1

TenAsys created the INtime OS to address the growing business in this area by more tightly coupling the iRMX real-time environment with the general-purpose Windows environment.

The INtime product was launched in 1997. What started in 1990 as DOS RMX (and iRMX for Windows in 1992) changed its API and application memory models so it was more conducive to using tools such as Visual Studio, the preferred development environment for Windows programmers. This transformed many operational aspects but retained, at the heart, the iRMX kernel. The evolution of iRMX to INtime involved enhancements to interprocessor communication, sharing of the I/O interrupts on a system, and changing the real-time OS APIs to provide a rich feature set of I/O and real-time OS capabilities while leaving the graphics and disk subsystems to the Windows environment. While iRMX does have a file I/O subsystem, it is not typically used in the INtime environment, relying instead on Windows for file and graphics I/O.

Intel multicore technology overview

The Intel multicore architecture is as it sounds, two processor cores on the same chip that share the same bus. Intel also provides an on-chip component called the SmartCache, a Level two cache that provides 2 MB of shared cache for both processors and arbitrates between the two. Multicore technology allows the processors to run at lower clock speeds, which reduces the power consumption of the part, another consideration for embedded applications as well as laptops and consumer devices. Other than those items, the part basically looks similar to two processors on a single chip. Memory and I/O are partitioned between the processors, and the traditional multiprocessor board architectures have been shrunk into a single chip.

Real-time and non-real-time processing

Embedded systems of today can typically be broken into two distinct subsystems:

1. The subsystem performing the real-time tasks of gathering information or performing operations that need to meet real-time deadlines
2. The subsystem performing the reporting, storing, and user interface tasks for the application

TenAsys takes the approach that INtime should be optimized for real-time computing tasks with a tight coupling and easy-to-use interfaces that allow Windows to do the information storage, graphics, and user interface displays where it is best suited.

The INtime real-time OS

It is interesting to note that as multiprocessor hardware has evolved from loosely coupled, multiple processors in a card cage to being on the same circuit board, and then finally, to today's Intel multicore processors, TenAsys has followed the same parallels with their real-time OS.

A typical installation model starts with a Windows environment installed in the standard way. Then the memory and I/O resources are partitioned between the two operating environments. Install the INtime real-time OS on the processor performing the real-time tasks for the system. Then bridge the INtime real-time OS with Windows using an NTX DLL on Windows that provides the communication path between the real-time and non-real-time environments.

Figure 2 shows how INtime provides the graphical ability to allocate devices to the real-time environment (INtime) or pass to the Windows environment. All the INtime configuration is done through graphical applications that run on Windows.

The real-time OS characteristics of the INtime real-time OS present typical real-time OS numbers:

- Less than 10 microseconds interrupt latency
- Multiprocess
- Multithreaded OS
- Real-time network stacks
- Drivers
- Interfaces

The INtime environment has capabilities to allow direct access to hardware I/O programming, which is often important to easy and efficient real-time programming. There are also independent C and C++ real-time-safe libraries. Exceptions are trapped and handled without causing a system crash.

Perhaps the most interesting aspect of the coupling between INtime and the Windows environment is the sharing of the hardware interrupts in the chip. INtime treats the CPU as two sockets. The latest Windows OSs have been written to run in a symmetric multiprocessor environment already, so they are ready for multi-core hardware. Hardware interrupts are managed by the *advanced programmable interrupt controller*. The controller includes an I/O controller, with each CPU having a local interrupt controller. The local interrupt controllers work with the main I/O controller to manage interrupts to and from each CPU. The INtime OS takes control of the overall interrupt controller and exposes a *virtual* controller to the Windows environment. During configuration, INtime then handles all interrupts that have been configured for the real-time side of the application and passes through the other interrupts for the non-real-time side of the application through this virtual interface. Additionally, INtime also takes control of the system clock tick in the same manner and exposes a virtual clock interface to the Windows OS.

The NTX layer of INtime does the interprocessor communication between INtime and Windows. It starts with a shared memory partition. Constructs are overlaid on this area to provide high-level constructs for mailboxes, pipes, single/multivalued

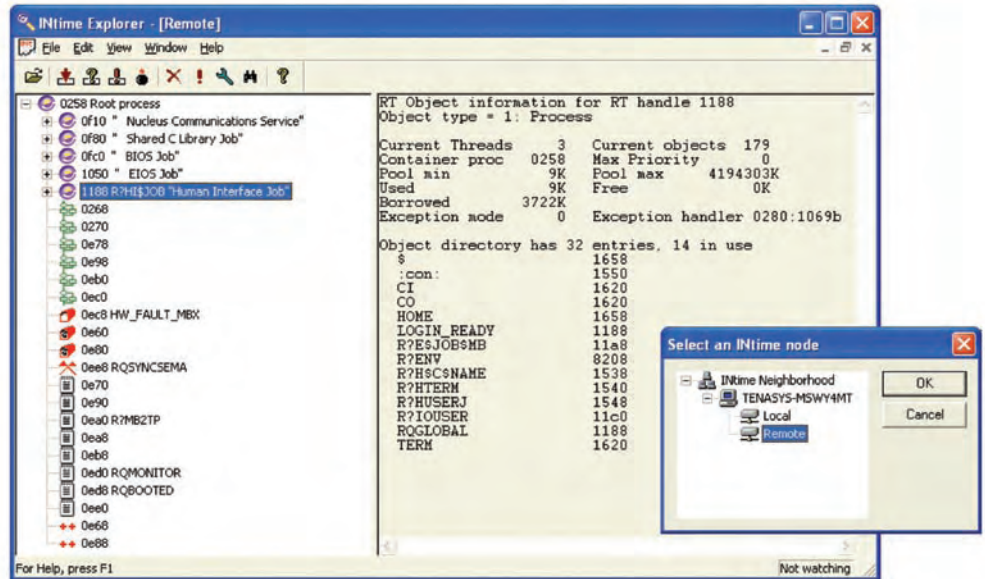


Figure 2

semaphores, physical shared memory, and mutexes. Signaling between the OS environments uses the interprocessor interrupt capability between the two cores. Blocking and nonblocking communication can be performed through these APIs.

File I/O is done through a pipe interface between Windows and INtime. Therefore, Windows controls the actual file reads and writes for the embedded application.

By coupling the real-time and non-real-time environments, questions may arise as to the impact of the real-time performance of INtime on a multicore processor if the Windows environment gets busy. During my visit with TenAsys, they showed me a demonstration where the INtime OS was performing a real-time task every 30 microseconds. Then various time-consuming tasks were launched on the Windows side. As the Windows side of the dual-core processor got busier and busier, the INtime real-time environment stayed rock solid at 30 microseconds, showing a surprisingly autonomous environment even with the coupling that had been done between the two OSs.

INtime development environment

Perhaps one of the best features of the INtime environment for embedded systems development is how well it is integrated with the standard Microsoft development tools. So, instead of carrying two development environments, the developer uses just one, Visual Studio, to program both the real-time and non-real-time sides of the application.

In addition, INtime enjoys the ability to provide a number of Windows-based diagnostics tools. A couple of these tools are INtime Explorer and INscope.

INtime Explorer is built upon the Windows Explorer paradigm and provides graphical access to all the devices, processes, and threads running within the INtime environment. The Explorer is organized by process space. Each process has a name space containing shared memory and/or objects, similar to how you look at the file system using Windows Explorer. Additionally, properties of these devices and the INtime environment are also accessible through this graphical application.

INScope provides a system-level analysis tool for debugging process, thread, and interrupt interactions (Figure 3). This graphical tool provides the ability to quickly diagnose and fix real-time applications involving sometimes complex interactions between threads, processes, and interrupts that occur during operation.

Developers can also set up markers on a variety of occurrences within the system using INScope to capture tough to debug real-time problems that occur infrequently in the system.

The Visual Studio environment has been extended where TenAsys has provided a plug-in for INtime. For those familiar with programming in Visual Studio, the developer simply does a:

create → *project* → *INtimeRTA* or *INtimeRSL* project.

INtime RTA class is a real-time user mode application. An INtime RSL class is a real-time shared library. The INtime application wizard has a full-featured GUI to create a graphical environment for specifying the resources used by the application. The result is a template in C or C++, and it also creates a project in Visual Studio. Visual Studio has the concept of a solution where multiple projects can be grouped. TenAsys uses this facility to wrap the developer's embedded solution into one Visual Studio solution containing Windows and INtime projects.

The Visual Studio debugger has also been integrated with INtime, so traps and exceptions invoke the debugger where you can perform single-stepping, insert breakpoints, and use other standard debugging constructs to debug real-time applications running in the INtime real-time OS environment.

Simulation of INtime applications also becomes a nonissue. Any of the new multicore PC environments can be used as an embedded INtime target machine. So instead of simulating on a sometimes inaccurate simulation environment, you can run the real software on the real OS on a real target, far better than hoping a simulation environment is accurate and up-to-date with the embedded OS you might be deploying.

Other non-real-time environments

So, what if the developer wants to use Linux on the non-real-time side of their application? Intel has been developing a concept with their multicore processor called *virtualization technology*. This technology enables the development of *hypervisor* operating software that can host guest operating OSs. Microsoft has been working toward support of this technology, as has TenAsys. TenAsys is adapting INtime to become a real-time hypervisor, working not only with Windows as it does today, but also able to host other OSs conforming to the interface. TenAsys' solution will interface to any other OS required for the application. Linux is one obvious choice.

AdvancedTCA and CompactPCI applications

What does all this have to do with CompactPCI and AdvancedTCA systems? Look no farther than AdvancedTCA target applications and architecture and see the common ground with INtime embed-

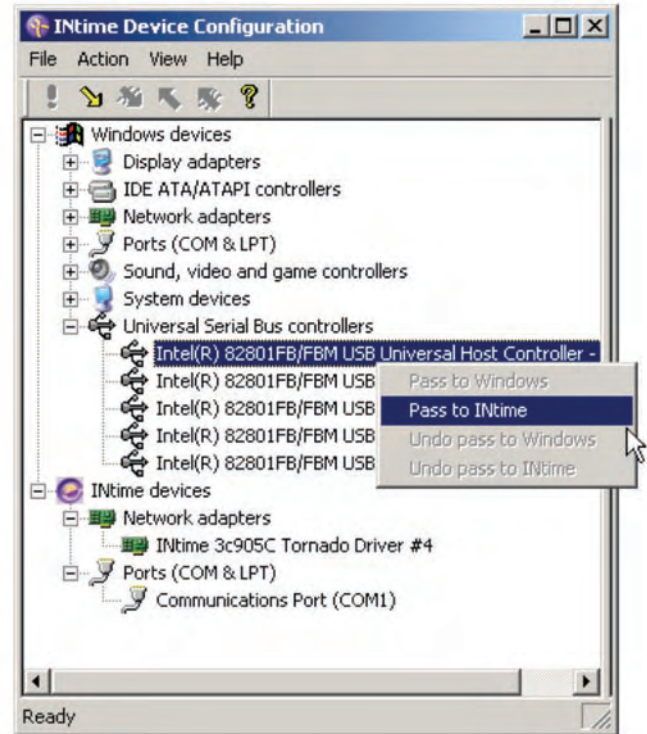


Figure 3

ded applications. Controller cards for AdvancedTCA systems will incorporate multicore processors. Further, the backplane of AdvancedTCA systems and communication with various other cards in the system may require real-time or non-real-time processing. A typical Voice over Internet Protocol (VoIP) application includes the real-time tasks of voice, data, and video stream processing coupled with user interface display. User interface display and input blocking that could impact the real-time performance of the system can be isolated using INtime to control the real-time aspects of the AdvancedTCA system. This kind of software architecture overlaid on the system also applies to areas such as industrial control, medical, and test and measurement systems.

Another interesting aspect to this is the AdvancedTCA architecture and the INtime history with Multibus II. The iRMX real-time OS ability to manage Multibus II systems with backplane message passing could make a reappearance in applications involving today's AdvancedTCA systems.

Conclusion

Most of today's embedded applications involve a dimension of non-real-time graphics and multimedia processing with a dimension of hard real-time processing that must be maintained in order for the system to work properly. Software solutions such as TenAsys' INtime address the need for combining hard real-time responsiveness with general-purpose processing and graphical interfaces with familiar, easy-to-use development tools to shorten design cycles.

For more information, contact Curt at cschwaderer@opensystems-publishing.com.