



Maximizing Performance with Intel® Multi-Core Architecture

By Jean-Charles Grabiaud of Ardence, Inc.

Intel® Multi-Core architectures provide obvious performance advantages over single-core and even multiprocessor configurations. But when combined with Ardence's deterministic extension to Windows (RTX), critical applications can be dedicated to an entire core, harnessing its full potential, with no interference from the operating system or other applications. Benefits include performance optimization with reduced latencies and improved reliability.

Challenges

Multiprocessing systems have been around for years, but since the operating frequency barrier was recently hit due to power consumption, Multi-Core architecture from Intel has been gaining popularity – and for good reason. Compared to conventional uniprocessor (UP) or multiprocessor (MP) chips, multi-core architectures can offer greater performance per watt, greater compute power through concurrency, greater system density and reduced thermal dissipation. While these hardware benefits are relatively straightforward, the true art is in the optimization of the system software that runs on these platforms and minimizing the complexity of the system design.

What about the existing code?

Parallelism is not new – software developers have been working with it for some time. But a lot of embedded companies simply do not have the required experience to develop it, and most of them have already invested in a code base created primarily for UP architectures. These companies need a solution to allow their code to achieve the greatest possible resource utilization on Multi-Core systems, with a minimal porting effort.

“Power is nothing without control”

Determinism is still a major challenge in embedded designs, and although improved performance can dramatically increase processing throughput and average response times, it won't make a non-deterministic system become deterministic, or even always improve worst-case response times. Some companies will identify their need as soft real-time, others hard real-time. Technically, it is either real time, or it is not.

Software design is critical to realizing multi-core performance. And software considerations start with the Operating System.

The need for “controlled” Windows

The popularity and market share of Microsoft® Windows® Operating Systems (OS) in industrial environments has dramatically grown in recent years. The reasons include:



- the industry-standard HMI and the wide variety of “in-place” functionalities
- the readily available experienced developers and the abundant existing applications
- the reliable support and the long-life roadmap from Microsoft

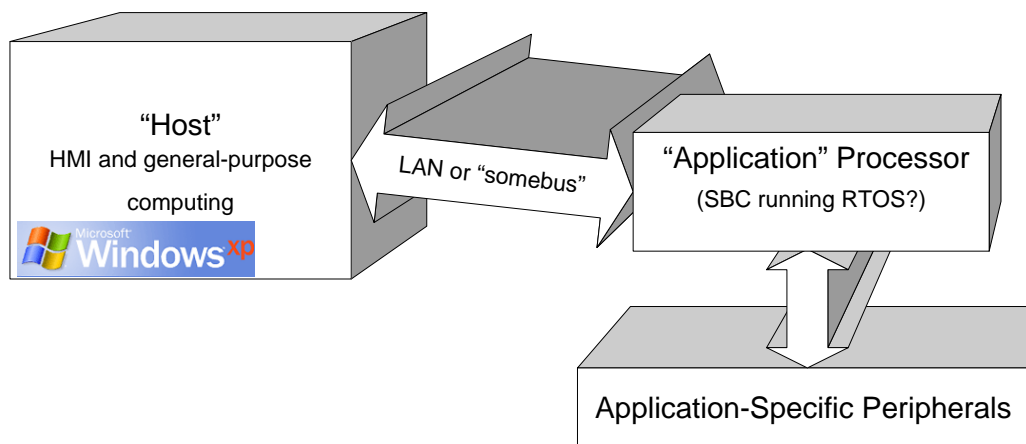
Because of the added complexity and cost of maintaining a heterogeneous computing environment, companies are striving to use Windows as their OS at all levels of an organization. Its use as a network server system or as a desktop system is easy to understand, since these are the very environments for which Windows was designed. However, there is also an impetus to use it in other environments, such as the factory floor, medical devices, and simulation, test, and communications equipment. A common characteristic of these environments is that they often require hard real-time system behavior.

Can Windows fulfill this need? The answer is... not out of the box.

The Microsoft Windows OS has been designed as a General-Purpose Operating System. The shortcomings in real-time applications have been thoroughly identified (too few threads priorities, opaque and non-deterministic scheduling decisions, priority inversion...) and leads us to these observations:

- it doesn't offer enough control to favor target applications as much as needed
- it doesn't keep critical application running if a fatal exception occurs
- developers are accustomed to easy access to hardware, and the Windows Driver Model is complicated
- Windows wasn't designed for managing “hard” real-time requirements

The time-honored solution...



Traditionally, developers would resort to adding a second real-time device to their system. But the obvious drawbacks are a higher hardware cost, a limited integration and higher cost, and the skills required for the development tools that differ between the “host” and the “application” processor.

And considering the current compute power available with Intel Multi-Core architectures, it's unfortunate that Windows does not utilize it effectively in the embedded space.

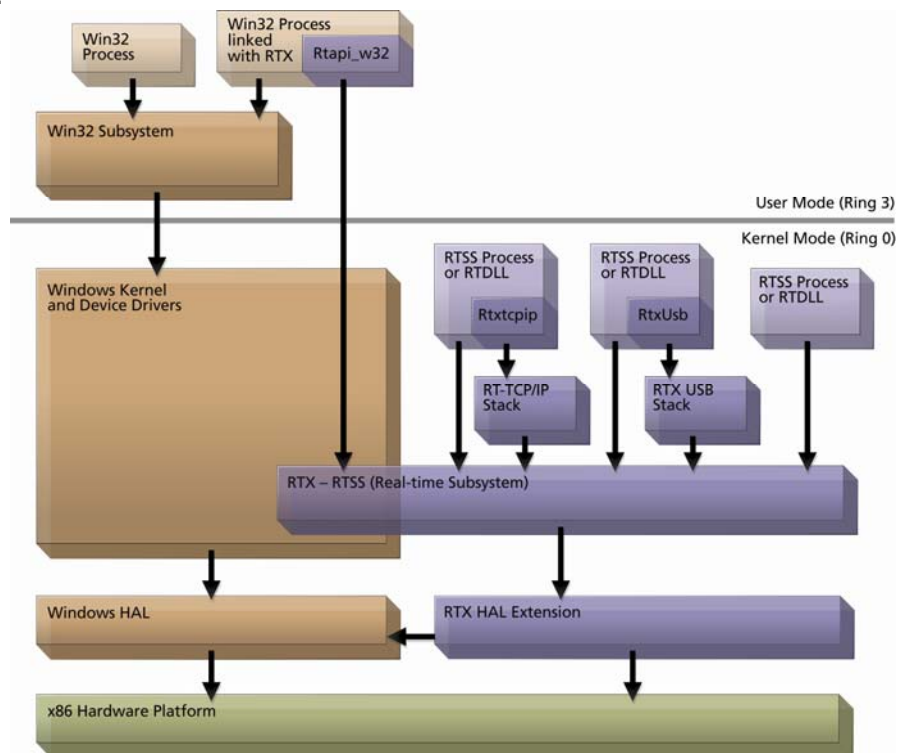
The demand

For embedded markets, the demand is for a cost-effective alternative to custom hardware (PC based, COTS OS), and the convenience of Windows, but with real-time attributes.

Ardence RTX solution

The Windows XP operating system is a mass-market product – not easily tweaked for niche applications such as real-time. The proper way to add real-time to Windows XP is via an extension, or a plug-in to the generic product. Ardence RTX is this real-time extension.

RTX is implemented as a collection of libraries (both static and dynamic), a real-time subsystem (RTSS) realized as a Windows XP kernel device driver, and an extended HAL (see Figure below). The subsystem implements the real-time objects and scheduler previously mentioned. The libraries provide access to the subsystem via a real-time API, known as RtWinAPI. RtWinAPI provides access to these objects. Note that the RtWinAPI is callable from the standard Win32 environment as well as from within RTSS. While using RtWinAPI from Win32 does not provide the determinism available within RTSS, it does allow much of the application development to be done in the friendlier Win32 programming environment rather than that provided by the DDK. All that is necessary to convert a Win32 program to an RTSS program is to re-link with a different set of libraries. The Windows XP Service Control Manager directly loads RTX process and DLL executable images—just as RTX itself—into kernel non-paged memory.



The key pieces of RTX include the HAL extension, the RTSS scheduler, the IPC mechanisms, the precise clocks and the development tools available.

The HAL is the one piece of the Microsoft Windows XP system for which the source is available for modification and extension. RTX has modified the HAL for three purposes:

- To add interrupt isolation between Windows XP and RTSS threads.
- To implement high-speed clocks and timers.
- To implement shutdown handlers.

In addition to interrupt management and fast-timer services, the real-time HAL also provides Windows XP shutdown management. An RTSS application can attach a Windows XP shutdown handler for cases where Windows XP performs an orderly shutdown, or crashes with the so-called Blue (Stop) screen. An orderly shutdown allows RTSS to continue unimpaired and resumes when all RTSS shutdown handlers return.

The RTSS scheduler implements a priority based preemptive policy with priority promotion to prevent priority inversion. The rule is: any RTSS thread can preempt any Windows thread. RTSS entirely eliminates latencies from IRQ masking by Windows XP and Windows XP drivers. The RT HAL performs interrupt isolation, reprogramming the PIC when switching between Windows XP and RTSS. The result is that RTX interrupts can always interrupt Windows XP, while RTX masks all Windows XP interrupts while RTSS is running.

Inter-environment IPC is a key feature of RTX, allowing tightly integrated applications where hard real-time processes run in the more resource-intensive RTSS environment, and the rest of the application runs in the Win32 subsystem. The IPC set includes mutexes, events, semaphores, and shared memory objects.

An important architectural feature of RTX is its lockless interrupt-driven interface between Windows XP and RTSS that implements a Local Procedure Call (LPC) mechanism. This clean architectural separation has enabled ports of RTSS to various environments (for example, multiprocessor RTX product) ensuring a fast and robust implementation.

RTX Clocks periods supported are 100, 200, 500 and 1000 microseconds. Because precise execution of events is critical in a real-time system, RTX provides three clocks on which to base event timers. Clock resolution, depending on the clock used, can be precise to within 0.001 nanoseconds – without any drift.

Microsoft Visual Studio IDE is fully supported. By providing a comprehensive suite of tools that integrate smoothly into it, software developers can significantly reduce development and debugging time.

These tools provide the ability to interactively view the application in real time to understand the interactions between hardware, RTX and the RTX application, to easily debug and analyze application behavior.

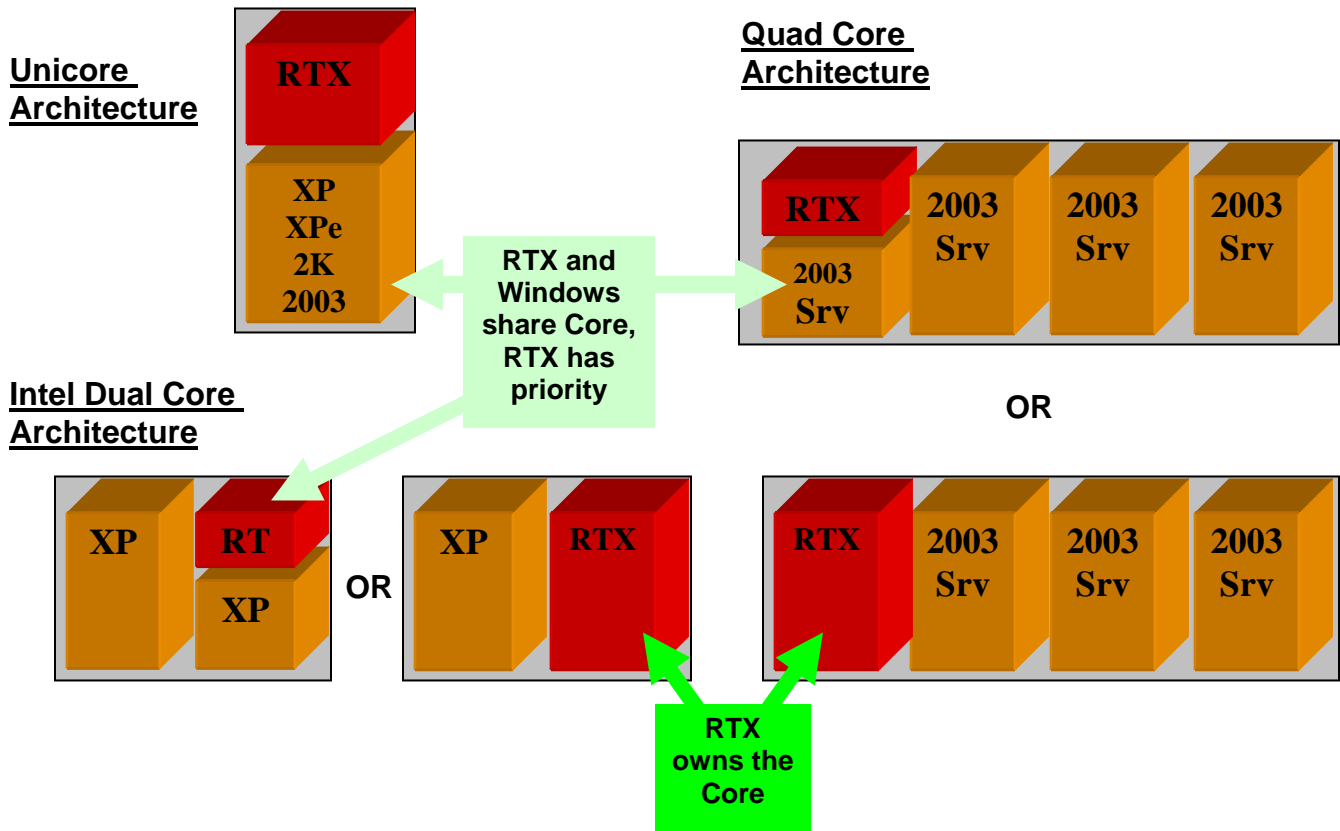
Ardence RTX for MP and Intel Multi-Core systems

Before 1999, RTX ran on single processor systems only. But for more than 7 years now, RTX releases run on multiprocessor systems and on Multi-Core systems as soon as they were introduced.

Intel Multiprocessor Specifications provide for interrupts to be controlled by an Advanced Programmable Interrupt Controller (APIC), suitable for a multiprocessor system. Through the APIC, different interrupts can be steered to different sets of processors.

On a multiprocessor system (MP), RTX can be configured in one of the following ways:

- Shared Mode - One processor handles both RTX and Windows processing; all other processors are dedicated to Windows processing.
- Dedicated Mode - RTSS dedicates one processor of the system to running RTSS threads, while the remaining processors run Microsoft Windows XP threads. This dramatically lessens the latency of real-time threads while preventing the processor starvation of Windows XP threads possible on a single processor system.





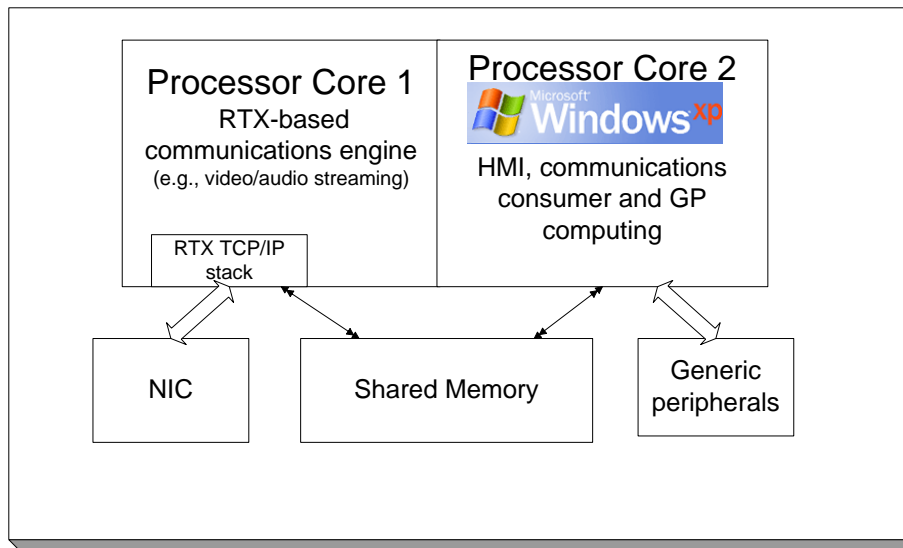
The optimal design...

- An Intel multi-core processor
PLUS
- Ardence's RTX operating in "dedicated core" mode
 - One core is dedicated to your application. Windows retains the remainder.
 - All your application processes take precedence over Windows processes using a separate scheduler.
 - Your application continues to run even if Windows encounters a fatal exception.
 - You can continue to use Windows multithreading along with RTX multithreading.
 - Windows and RTX processes communicate through shared memory.
 - Your application can access hardware without going through the Windows driver model.
 - You can develop using Visual Studio for both Windows and RTX.
 - RTX offers "hard", real-time performance if required.

Ardence RTX "Dedicated Core" mode typical applications

Dedicated communications engine

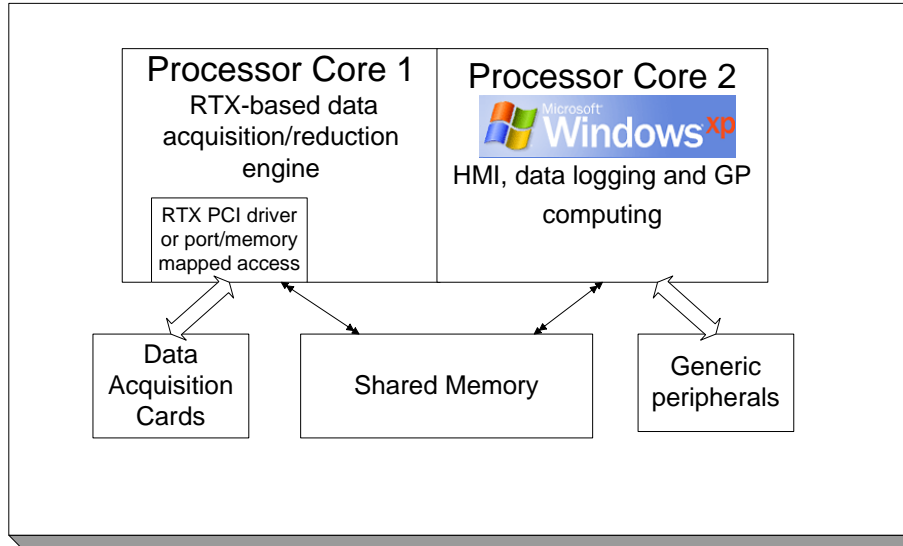
RTX can handle all the video/audio streaming through its own stack on a single, dedicated core, while Windows completes the remaining tasks on its own core as well. Communication between the two is efficiently realized thanks to the IPC mechanisms that RTX provides.





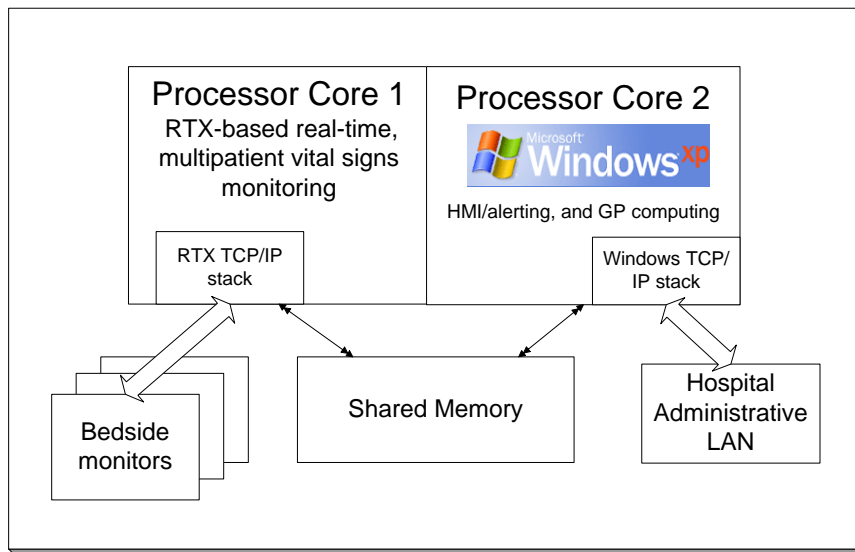
Data acquisition/reduction engine

In the example below, very common in the Industrial Automation market, the acquisition of critical data or the control of remote devices through a PCI interface entirely handled by RTX, leverages the full power of a core, minimizing latencies and allowing even more complex processing on the acquired data. Windows, on the other core(s), won't interfere in these tasks but won't be starved either.



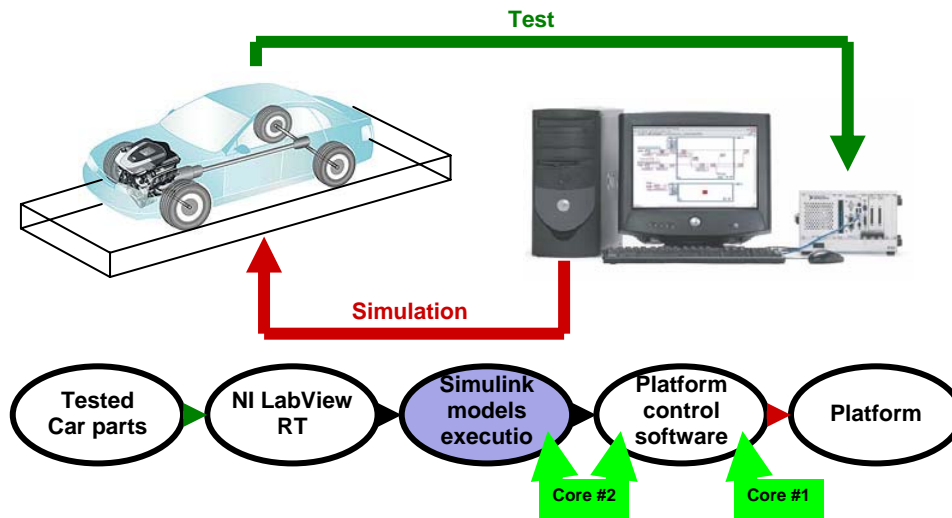
Multi-patient vitals monitoring

Medical applications are also numerous, but this example focuses on a monitoring system where a dedicated network is fully handled by RTX through an isolated network interface. The data are obviously available for Windows, and any application on the hospital LAN can retrieve and use them as desired without interfering with the monitoring.



Automotive Simulator: Test and Simulation on a single system

This last profile is becoming more and more common in the automotive market where the system providers must differentiate by offering both test and real-time simulation capabilities on a single system, while reducing the overall cost of their solution. MathWorks products are often used in this context and RTX can handle the very complex simulation models in dedicated mode, preventing additional hardware and integration costs of a secondary DSP-like device



About Ardence

Ardence develops software platforms for the on-demand world. Our Software-Streaming Platform enables IT to become more agile by delivering a virtualized IT infrastructure. Our Embedded OEM Development Platform Enables Embedded OEMs to leverage Ardence’s control and real-time technologies – and our Embedded Windows expertise – to develop systems that deliver increased performance and manageability.

Founded in 1980, Ardence is headquartered in Waltham, Massachusetts, and operates across North America, Europe and Asia. Ardence is a General Member of the Intel® Communications Alliance. More information about Ardence Inc. can be found at www.ardence.com.



NORTH AMERICA
 266 2nd Avenue
 Waltham, MA 02451
 Toll-free: (800) 334-8649
 Fax: (781) 647-3999
 E-mail:
 EmbeddedSales@Ardence.com

EUROPE
 ABS – Porte de l’Arenas, Hall C
 455 Promenade des Anglais
 06299 Nice Cedex 3- France
 Tel.: + 33 (0)4 89 06 60 10
 Fax: + 33 (0)4 89 06 60 20
 E-mail: fboisset@Ardence.com