RTX transforms Windows into a Real-Time Operating System

For complex, connected embedded systems that want to take advantage of the Windows HMI and also require determinism and hard-real time, RTX provides a real-time operating system scheduler (RTOS) that is tightly integrated with Windows.

Embedded systems developers worldwide in industries ranging from Industrial Automation, to Medical Systems, to Digital Media are standardizing on Microsoft's Intelligent Systems Architecture. The Intelligent Systems Architecture's powerful Windows user experience, the Visual Studio development tools, and the unmatched support structure of the Microsoft Developer Network (MSDN) deliver measurable cost, performance and scalability benefits. Increasingly, OEMs and end users with stringent performance criteria for their hard real-time embedded systems rely on IntervalZero's RTX hard real-time software to augment Windows capabilities.

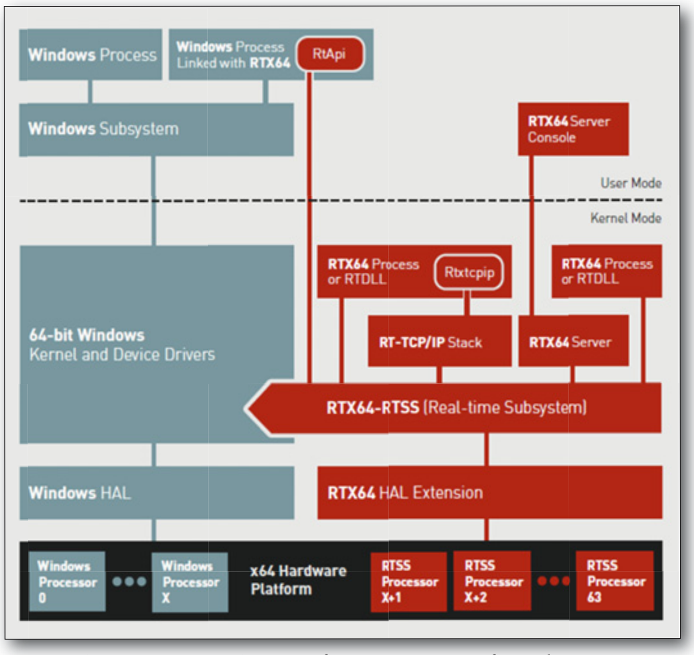
For complex, connected embedded systems that want to take advantage of the Windows HMI and also require determinism and hard-real time, RTX provides a real-time operating system scheduler (RTOS) that is tightly integrated with Windows. In short, RTX transforms the Windows general-purpose operating system into an (RTOS). Where Windows provides timers with a maximum resolution – smallest granularity – of 1000 µs (1 millisecond), RTX lowers this to 1 µs where supported by the hardware. RTX extends the Windows operating system's capabilities – without altering or modifying the Windows Hardware Abstraction Layer (HAL) – to deliver determinism and hard real time performance on both 32-bit and 64-bit systems. RTX software does not rely on latency-inherent virtualization approaches or unnecessarily complicated interprocess communications schemes.

Uniquely, the RTX64 RTOS scheduler enables embedded real-time applications to directly access the 512GB of addressable physical memory available on 64-bit Windows. This is critical to modern day real-time systems and represents a gigantic leap from the 4GB physical memory limit of traditional 32-bit Windows systems. The 4GB barrier has stymied innovation in many industries that depend on real-time systems and that require memory access far beyond 4GB. Industries where the operations console will benefit from a bigger field of view are obvious places where 64-bit can be applied.

Medical imaging, simulation, along with video and sound mixing top the list of applications that will benefit significantly. All have large memory requirements today that exceed the addressable memory capabilities of 32-bit computers. Having access to 512GB of memory frees machine builders in these key industries to develop more sophisticated, break-through systems. And 64-bit also creates opportunities for competitive separation in industrial automation – CNC, machining and other industrial systems. Thanks to the larger addressable memory, vision systems can be added to traditional machine controllers at a very low cost. Vision systems assist in the positioning of cutting tools, improving both quality and speed.With its native 64-bit RTOS scheduler RTX64 joins with 64-bit Windows to create a powerful, efficient, and integrated platform for today's real-time systems and tomorrow's. The Windows HMI is used in many embedded systems today that also rely on a separate RTOS, or on proprietary hardware such as DSPs to provide real-time functionality. By adding RTX, a single operating system – Windows – with a single codes base, and a single development team can deliver both a world-class user interface and the required real-time capability. Additionally, developers can harness RTX's symmetric multiprocessing (SMP) capabilities to supplant DSPs and achieve breakthrough scalability on today's multicore multiprocessors, taking advantage of Moore's Law and doubling performance every 18 months.

In today's struggling economy, the complexity and cost of maintaining heterogeneous computing environments are catalysts for companies to standardize on the Windows architecture at all levels of their organization – including environments that require hard real-time system behavior.

An overview of the RTX architecture and more detailed technical information on some key aspects of RTX's functionality will help developers and product leaders understand both how it works and the value of this real-time extension of Windows.



IntervalZero RTX64 real-time software extension of Windows architecture

The RTX Real-time Scheduler

RTX implements a single symmetric multiprocessing scheduler across all of the cores in the real-time subsystem. Instead of a separate RTOS and application running on each real-time core, RTX uses a single real-time scheduler to schedule threads across a number of different cores. This flexible SMP scheduling model makes synchronization and communications among the different cores/ threads straightforward and easy to implement. Load balancing is implemented from the APIs provided by the RTSS. The RTX APIs enable threads to be moved between processors during runtime. This scalability translates well when moving the application to systems with different core configurations. When cores are either added or removed from a system, the user can use logic within their application to load balance threads as needed. Of course every embedded system is different and has a different set of requirements. RTX provides the flexibility to configure the combined Windows/RTX implementation to best optimize the system for the intended purpose.

The RTSS scheduler has been coded with the requirements of real-time processing in mind. Most importantly, its operation is low latency, and is unaffected by the number of threads it is managing. Each priority has its own ready queue, maintained as a doubly linked list. This allows the execution time of insertion (at the end of the list) and removal (from anywhere in the list) to be independent of the number of threads on the list. A bit array keeps track of which lists are non-empty, and manipulating this bit array is done by high-speed assembly-coded routines. As detailed a bit further on below, while an RTSS thread is running, all Windows-managed interrupts, as well as any interrupts managed by threads of a lower priority than the current thread, are masked out.

Symmetric Multiprocessing Configurations

SMP configurations require a system with multiple processors/cores that can assign 2 to n processors/cores (where is n is 32 or 64 depending on whether RTX32 or RTX64 is being used) exclusively to the real-time subsystem. RTX SMP Runtime also supports non-MP configurations. RTX with SMP provides for seamless execution of real-time threads across processors via a single RTSS instance. Again, this is very important in that there is little to no resource-usage redundancy as is the case with an asymmetric multiprocessing (AMP/ASMP) design.

The RTX SMP implementation is optimized for embedded systems. RTX provides mechanisms such as the following, to help embedded developers create systems that meet their design requirements including: explicit thread migration; explicit thread/process level processor affinity; IRQ/ISR/IST level processor affinity for deterministic device I/O; accurate thread timing API for effective load balancing; highly optimized interprocess communication synchronization objects; and highly efficient direct access to (non-buffered) shared memory.

Real-time Hardware Abstraction Layer (HAL)

The RTX HAL Extension module performs dynamic HAL detection in calls related to memory – interrupts, timer, shutdown – and redirects them to their RTX counterparts. This binary hooking technique has a number of advantages over HAL replacement:

- RTX handles a broader range of OEM platforms, as call redirection is limited to calls that vary little among different OEMs and service providers;
- installation becomes more robust, as the on-disk copy of the HAL is untouched, and RTX is generally unaffected by Windows hot fixes and service packs.

The Windows HAL is extended by RTX for three purposes: to add interrupt isolation between Windows and RTSS threads; to implement high-speed clocks and timers; and to implement shutdown handlers. Interrupt isolation means it impossible for a Windows thread or a Windows-managed device to interrupt the RTSS. It is also impossible for a Windows thread to mask an RTSS-managed device. The RTX Hal Extension ensures that these conditions are met by controlling the processor's interrupt mask. When running an RTSS thread, all Windows-controlled interrupts are masked out. When a Windows thread calls to set the interrupt mask, the RTX Hal Extension, which is the software that actually manipulates the mask, ensures that no RTSS-controlled interrupt is masked out.

Relative to clocks and timers, as noted earlier, Windows provides timers with a maximum resolution (i.e., smallest granularity) of 1000 µs (1 millisecond). The RTX Hal Extension lowers this to 1 µs where supported by the hardware.

The RTX HAL Extension also provides Windows shutdown management. An RTSS application can attach a shutdown handler for cases where Windows performs an orderly shutdown, or halts with a bug check. An orderly shutdown allows RTSS to continue unimpaired and resumes when all RTSS shutdown handlers return. In a bug check case, RTSS shutdown handlers run with certain limitations. In practice, it means that a shutdown handler should clean up and reset any hardware state, possibly alert an operator, or switch to a hot spare when the system stops because of to either a normal shutdown or a crash.

Memory Management

Memory Allocation - RTX uses the Windows memory allocator for all of its memory allocation. All memory that is allocated to the RTSS subsystem is from the Windows non-paged pool. Because RTX must request memory from Windows, memory allocation is not deterministic. In cases were memory allocation needs to be deterministic, RTX supports a local memory pool,

Local Memory Pool - Once the local memory pool has been created, all memory allocations that use it are managed by RTSS and are deterministic. It is recommended that the memory usage for a given system be characterized in order to optimize the subsystem local pool size before the real-time processing of the applications starts. This ensures that future memory allocations do not cause more round-trip requests to Windows to grow the pool size.

RTX Interrupt Handling

The RTX HAL Extension works with the Windows HAL to manage interrupts in order to insure that RTX interrupt priority is higher than Windows. Because the RTX interrupt is at a higher priority than Windows interrupts, the RTX interrupts will always be available regardless of what Windows is doing at any time. RTX interrupts include the RTX HAL clock tick RTSS interrupt devices (line-based or message-based), and Service Request Interrupts (SRI).

Message Signaled Interrupts - PCI version 2.2 introduced a new interrupt architecture based on message signaling rather than the standard line-based signaling. This architecture does not rely on hardware lines and does not have the limitations often associated with running out of resources or inflexibility in configuration of system peripherals. Message signaled interrupts (MSI) uses the systems memory to write interrupt messages directly to the interrupt controller. With RTX's support of MSI, embedded designers can take advantage of MSI. This is a huge advantage when dealing with situations where Windows has ownership of IRQ lines that are needed by a real-time device. Selecting MSI capable devices resolves the conflict without any additional requirements other than the standard conversion process involved in moving a device from Windows to RTX.

RTX and Software/Hardware Interrupt Latency

A switch from Windows to RTSS happens on an interrupt, either from the RTX high-speed clock, or from another device generating RTX interrupts. Therefore, achieving RTX ISR determinism requires reducing Windows interrupt latency. The most significant is IRQ masking by the Windows kernel and drivers, routinely done for periods up to several milliseconds via Windows KeRaise/LowerIrql calls. The Windows kernel, HAL, and certain special drivers also perform processor-level masking of all interrupts via x86 STI/CLI instructions, for periods of up to 10 µsec.

Windows and RTX interrupt processing naturally masks interrupts, thereby adding to ISR latency. Although Windows relies very heavily on interrupts in many situations (e.g., raising software exceptions, or unwinding a thread's stack), the Windows interrupt sequence is reasonably compact in its contribution to the worst-case ISR latency.

The most obvious hardware-related problem is cache dirtying and flushing by applications and the operating system. This category also includes re-filling of the Translation Lookaside Buffers (TLBs) Video drivers are particularly aggressive users of caches, causing contention-related flushes when an RTX interrupts starts running. Histograms of ISR behavior in the presence of cache-dirtying applications would typically have a double-hump profile; with most samples near the best-case band, and another large number of samples in the flushing-related band.

Power management, especially on portable devices, creates occasional long-latency events when the CPU is put in a low-power-consumption state after a period of inactivity. Such problems are usually quite easy to detect. A typical system can disable those features via BIOS setup. Also Intel's® SpeedStep® Technology can cause long latencies when switching between power states. Some systems use the Pentium processor's System Management Mode (SMM) to perform specialized keystroke or other processing in BIOS firmware. While in SMM, the processor does not field interrupts, which adds to ISR latencies.

Bus mastering events can cause long-latency CPU stalls. Such cases include high-performance DMA SCSI devices, causing CPU stalls for periods of many microseconds; or video cards that insert wait cycles on the bus in response to a CPU access. Sometimes the behavior of such peripherals can be controlled from the driver, trading off throughput for lower latency. While no operating system can protect an application against such hardware factors, RTX offers tools to diagnose platform-related latencies, and identify the misbehaving peripherals. Being mindful of such factors and using RTX tools to qualify one's development platform are essential for a system's overall performance. ■

Massachusetts-based IntervalZero, Inc. has additional offices in France, Germany and Taiwan. RTX has a global customer base of market-leading OEMs and end users in Industrial Automation, Medical Systems, Digital Media, Test & Measurement, Military & Aerospace, and other industries. IntervalZero is a Windows Embedded Gold Partner and has been recognized each of the last four years with Windows Embedded Partner awards.