Transform Windows to Deliver Software-Only Real-Time Performance for IoT & Industry 4.0

IntervalZero

Next-generation industrial, vision, medical and other systems seek to combine high-end graphics and rich user interfaces with hard real-time performance, prioritization and precision.Today's industrial PCs running 64-bit Windows, complemented by a separate scheduler on multicore multiprocessors, can deliver that precise real-time performance on software-defined peripherals.

The pace of advancement in silicon integration and performance continues unabated, driven by the demands of ever-richer applications. That advancement, however, has been taking different directions, especially in the case of the integration of functionality onto single silicon dies. The task for the embedded OEM is to decide how to turn these revolutionary developments into real products that can provide what customers demand. That now includes high-definition audio and video, machine vision, real-time industrial products such as six-axis motion control, real-time connectivity and a rich user interface. That user interface often must also include the ability to present complex real-time graphical data that is linked to the application in real-time.

Today's hardware has that capability, but the way to realize it is through software. With processor cores that are powerful enough, there is no need to rely on custom hardware to implement specialized functions; that can now be done with software. Software can be more easily updated and improved than hardware, and it is here where OEMs can implement their real value.

The best way to smooth the task of implementing complex real-time software applications is to start with the right hardware environment. As noted above, hardware integration has been taking different directions. On the one hand, there is a trend to integrate the kinds of devices used by real-time systems onto a single die. These might include a multicore processor, a DSP, an FPGA or an advanced graphics unit. We have recently seen, for example, devices that integrate processor cores and FPGAs, or processor cores with advanced graphics units that are also capable of intense number computation such as DSP.

On the other hand, there is the opportunity to tap into today's multi-core CPUs, whose tremendous power and performance is the result of multiple cores and, to a smaller extent, clock speeds approaching 3 GHz. These standard commercial off-the-shelf industrial PCs (IPCs) provide platforms that, with some additional instructions and a scheduler, can deliver DSP-level processing, performance, prioritization and precision. With today's CPUs, this processing can be done in floating point for more diverse calculations than with the fixed point typically found in a DSP. Such performance changes the focus from trying to optimize the use of every instruction to actually fully exploiting the real power of the multicore IPC.

This trend has already resulted in devices that can outstrip traditional DSP processors. Another major development in this arena is the move to 64-bit architectures that are backward compatible with their 32-bit predecessors, but which offer enormously enhanced performance. This has several advantages, because even a highly integrated chip with different integrated functions with their different instruction sets and protocols throws up obstacles to a unified software environment, which adds both hardware hurdles as well as burdens on the development team to circumvent them.

The all-new implementation of IntervalZero's RTX64 takes the latter path and transforms Windows into a fully functional real-time operating system that runs entirely on x64 multicore hardware. Additionally, in so doing it provides access to 128 Gbytes of non-paged memory, depending on actual mapped physical RAM size. Overall, Windows' 512 Gbytes of physical memory dwarfs the 4 Gbytes physical

memory limitation of 32-bit Windows. This vast amount of available memory opens the door to previously unavailable applications like MRI medical imaging and high-end video editing to name a few.

Above all, RTX64 provides a single commodity hardware environment in the form of multicore x64 devices. This enables a single software environment that can accommodate Windows with its rich user interface, available applications and development environment. And Windows is seamlessly connected to the full-function real-time symmetric multiprocessing (SMP) RTX64 environment that can scale from 1 to 63 cores. Applications compile to a single code base with no need for FPGAs or DSPs to execute logic based on different code that must be separately compiled and linked with the main application. One set of hardware, one operating system environment, one set of tools and one base of code. That translates to one team that can communicate and work together and produce high-performance, scalable applications while dramatically shortening time-to-market.

A Liberating Unified Architecture

Freed from the isolation of the real-time system and from other functions such as the user interface, OEMs are able to explore more innovative solutions with less risk and overhead. For example, consider the user interface. Today's advanced applications and their users—are demanding feature-rich, interactive, touch-activated graphical user interfaces. Advanced embedded systems are far beyond the headless systems of yesteryear. In addition, it is becoming increasingly important to provide a definitive user experience based on that interface, one that can help reinforce product branding.

The old way of developing an embedded user interface was to have a team design the UI on a desktop system using a graphics program such as Photoshop, Illustrator, or perhaps one of the later tools aimed at embedded systems. The result would be a prototype UI with simulated data and interfaces. That would then be handed off to the team developing the real-time application, and their task would be to implement the UI design under whichever RTOS environment they were using. This inevitably entailed changes and compromises, and testing that was mostly put off until late in the development cycle with predictable effects on time-to-market.

Using RTX64, development teams can build their UI with whichever Windows-based tools they choose and be confident that the objects in the interface can communicate directly with the RTX64 APIs, and exchange commands and data with the embedded application. Any changes to the UI or to the embedded functionality can be quickly traced and updated between the two. The same seamless access to networking, databases and storage can be provided for hard real-time applications under RTX64 because it does not alter Windows in any way, but is a real-time extension to Windows. Thus its communications with Windows are seamlessly integrated and do not depend on mechanisms like remote procedure calls, virtualization, or hypervisors that are associated with other implementations of multiple operating systems. The addition of a second scheduler and an RTOS infrastructure allows UI functions to execute in Windows while real-time functions execute on the RTX scheduler. Putting the right task on the right scheduler delivers the best overall result.

RTX64—A Fresh Start into 64-Bit

The new RTX64 was built from the ground up to open the world of 64-bit real-time computing, and it is not a port of the 32-bit product. Professional audio and video, high-end medical devices along with advanced industrial control systems that incorporate machine vision and rich user interfaces, all place demands that can only be met by advanced 64-bit systems that can include the rich user interface possible

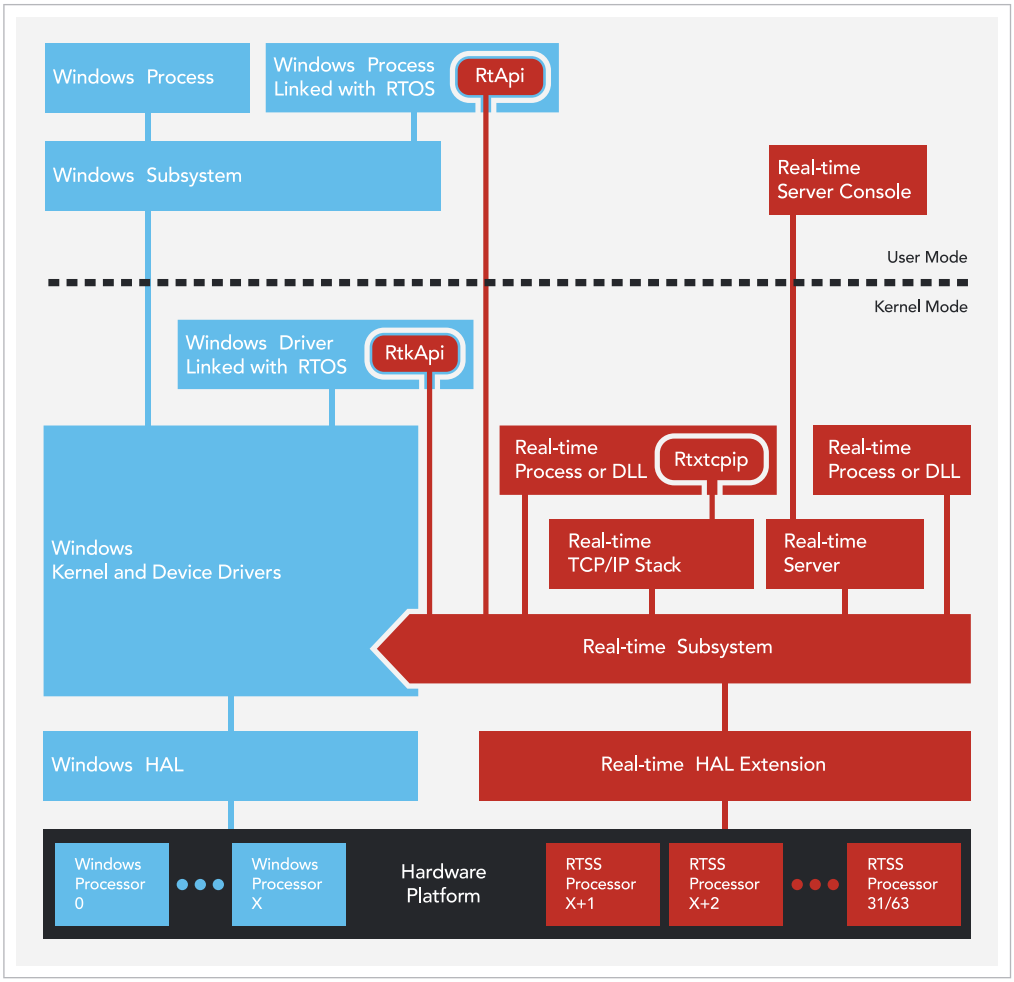


Figure 1

RTX64 provides an architecture that takes advantage of the advancing technologies—specifically, high-speed, multicore x64—that can outperform and outscale the traditional embedded environment that relies on DSPs, FPGAs and microcontrollers.

RTX64 provides an architecture that takes advantage of the advancing technologies—specifically, high-speed, multicore x64—that can outperform and outscale the traditional embedded environment that relies on DSPs, FPGAs and microcontrollers (Figure 1). It does this by implementing their functions at even higher performance in a single hardware environment, and it can do this in conjunction with Windows, which offers the rich user environment and access to a huge number of applications that can take advantage of and support the real-time operations.

To start with, RTX64 has a hardware abstraction layer (HAL) that is distinct from the Windows HAL, but operates alongside it. Thus, from the start, no modification of Windows is needed. The two systems operate side-by-side and communicate via existing mechanisms. The RTX64 HAL can scale from 1 to 63 cores to deliver deterministic real-time performance with timing down to 1 µs (dependent on hardware support). The scheduler, which resides in the RTX64 real-time subsystem (RTSS), can assign threads to cores to achieve symmetrical multiprocessing (SMP) withou relying on virtualization or complex interprocess communications.

This is also a result of the vast memory space that is available to all cores without memory partitioning. Up to 128 Gbytes of non-paged memory and up to 512 Gbytes of physical memory can be accessed by the entire system. This is a huge advantage for medical applications that increasingly depend on visualization such as the Optical Coherence Tomography (OCT) technology now under development, or for real-time surgical robots that depend on accurate rendering and processing of organ images like a beating heart. It is essential for advanced industrial control systems that must not only present visual data to the user, but also process it in real-time to drive motion control of tools and also for the inspection of parts produced by the process.

Having a memory space like this available to such a high-performance general-purpose hardware platform allows OEMs to develop specialized software that can perform extremely specialized functions that would have otherwise required specialized hardware components. Experience has shown that mixing different hardware involves quite different sets of software that depend on different disciplines (e.g., C++ vs. Verilog), which not only greatly slows development time, but also places limits on performance and scalability. Scaling such systems only brings increased complexity with each disparate piece of additional hardware with its own interfaces and unique software needs.

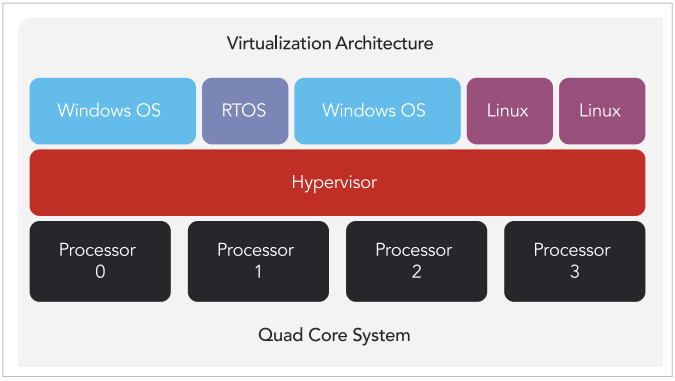
The RTX64 real-time subsystem (RTSS), which includes a real-time scheduler, is fully independent from the Windows kernel and the Windows scheduler. There is no inherent interaction or interference of Windows and real-time threads. Only intended communications between threads by the developer can occur using the real-time API. A real-time API is provided for use with user-mode Windows applications, or a real-time kernel API for use with Windows kernel drivers.

In scenarios such as those enabled by RTX64, applications can present themselves to the user as common Windows applications, while behind the user interface, many of their features are taking advantage of RTX64 real-time processes. For example, a machine tool control program might present a view of the part being machined along with controls and settings that the user can access via a touch screen. However, the actual application consists of two parts. The Windows program can communicate with the real-time control program on two levels—the kernel and the user level—by means of real-time APIs.

At the kernel level, a Windows driver can send data to the RTX64 side, which is perhaps controlling the travel of a tool, and receive current position data, which it then passes to the user interface, or subjects to some sort of processing via a real-time kernel API (RtkApi). At the user level, the operator can set values or the position of switches, etc., on the touch screen and these will communicate with a Windows process. That process in turn uses the real-time API (RtApi) to communicate with the RTSS. These two classes of API communicate directly with the RTSS, which is where the real-time control program resides.

As the demand for rich user interfaces for real-time and embedded systems continues to grow, developers are being faced with the dilemma of how to link such interfaces with RTOS environments that are traditionally not designed to support complex user interfaces. We have already mentioned the often-awkward tricks that must be performed to match such an RTOS-based system to a complex user interface. With the RTX64 extension to Windows, it is straightforward to use one's favorite graphical tools to design a user interface that can link directly to the underlying real-time application using the RTX APIs. Even more attractive to some could be the ability to simply purchase an off-the-shelf software control and data acquisition (SCADA) tool, which comes with many pre-designed but customizable gauges, sliders, switches and representations of pumps, tanks, actuators, etc., and develop from there using the same RTX64 APIs to hook up to the system.

The same goes for video data. There is a wide selection of tools and applications that can represent physical phenomena, such as heat distribution, fluid dynamics, stress and more, and they all run under Windows. Image processing applications exist that can do edge detection and other operations needed for parts inspection. The list goes on. The OEM has, at this level of the Windows user interface, a rich selection of "build or buy" options, all of which he can confidently use and/or experiment with knowing that the interface to the underlying real-time application is well-defined and will work out of the box.



RSMP Paves the Way to Performance and Scalability

There are, of course, different schools of thought on how to take advantage of multicore processors. These basically break down into asymmetrical multiprocessing (AMP), or virtualization and symmetrical multiprocessing (SMP). One approach to AMP requires that a copy of the operating system run on each of the cores. This then requires assignment of memory to the individual cores and brings with it the need for interprocess communications that add to overhead. If one tries to implement a user interface with Windows the same inefficiencies apply, requiring interprocess communications between Windows and multiple instantiations of RTOSs and memory partitions. Then try processing (under the RTOS) and displaying (under Windows) real-time video data in such a system—involving more IPC—and things clog up very quickly. Scaling the system to more cores requires more copies of the RTOS, more memory partitioning and reconfiguration of the application.

Another approach to AMP is to implement virtualization with a hypervisor, which is a separate layer of software running directly on the hardware that divides the hardware among the operating systems (Figure 2). Some multicore processors even have built-in hardware assistance for virtualization, which basically presents a virtual "motherboard" to each operating system. Virtualization is often used to support "separation kernels," which are isolated from the rest of the system, communicating only via tightly controlled mechanisms and protocols. This can be useful in certain cases, but its goal is isolation, whereas the goal of SMP is integration.

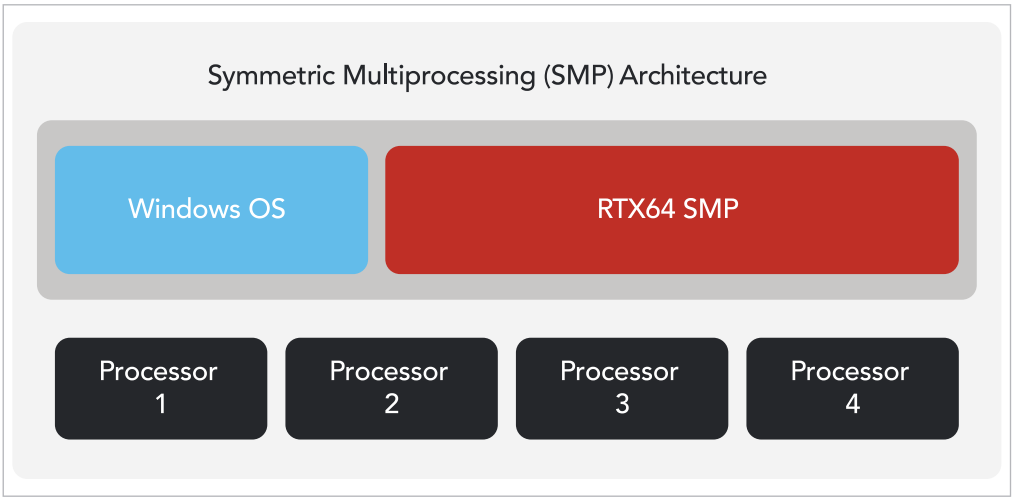
Figure 2

Another approach to AMP is to implement virtualization with a hypervisor, which is a separate layer of software running directly on the hardware that divides the hardware among the operating systems.

RTX64 represents a real-time operating system extension to Windows and works with Windows as a single operating system environment that uses the SMP approach to treat the multiprocessor hardware as a single shared resource. It requires only a single copy of the entire operating system environment including the real-time subsystem with its real-time scheduler that has access to all cores assigned to the subsystem (Figure 3). Unlike with AMP, the code can be written once and can be later scaled as functions are added by statically reassigning threads or adding cores and repartitioning. Since all the cores, and hence all the threads, have direct access to shared data and all resources are visible to all real-time processes, there is no need for additional copies or the use of complex interprocess communications schemes or remote procedure calls.

Figure 3

RTX64 represents a real-time operating system extension to Windows and works with Windows as a single operating system environment that uses the SMP approach to treat the multiprocessor hardware as a single shared resource.

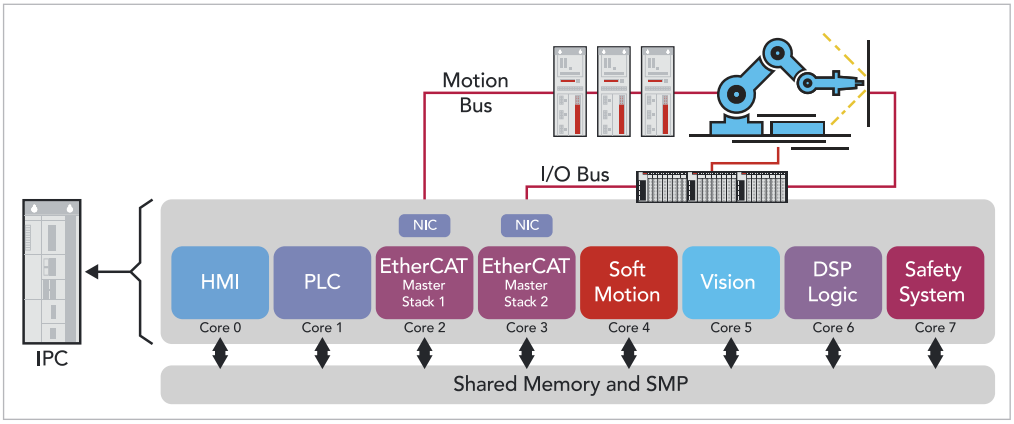


The ability to use a single extended operating system environment across a homogenous hardware platform reduces the OEM's major hardware decision to, "Do I have enough cores to do what I need to do?" or, "How many more cores do I need to add in order to scale this application to the additional functionality I need?" It no longer involves bridging interfaces between disparate hardware elements like FPGAs and DSPs, or adapting code to parts with increased performance but different programming needs. It no longer involves bringing in different hardware specialists to create or upgrade a product. The team defines the performance in terms of a single programming language like C++.

This leads to the additional advantage of having a single set of development tools, such as Windows Visual Studio, for the entire project. Windows serves as the development environment for the entire system—Windows functions as well as real-time coding. Other Windows-based tools can be brought into the mix as well, such as requirements analysis, version control or static analysis tools to name a few. The user mode of the real-time subsystem also includes an RTX64 server console that connects to the RTSS. The real-time crew can also use their favorite real-time debuggers, profilers and analyzers to tweak the real-time subsystem. They can all communicate and consult with each other in the same terms. Nobody has to learn Verilog or a DSP coding language.

Figure 4

EtherCAT provides for gateways to integrate existing fieldbus components such as CANopen or Profibus. EtherCAT runs under RTX64 in software without the need for any specialized EtherCAT card plugged into the system bus.



Connectivity—Internet and Real Time

With the rise of the Internet of Things, connectivity has become a must have in terms of linking devices to local networks, then to servers and ultimately to the Cloud. Internet connectivity is simply a given with Windows, and it can be customized to exchange data and commands with the real-time processes as well as provide for a remote user interface for interacting with the systems from virtually anywhere. However, the Windows Internet connection itself is not real-time.

Yet with the same systems running Windows and the RTX64 real-time extension environment, it is possible to add real-time Ethernet connectivity in the form of EtherCAT, which is an Ethernet-based fieldbus system for control automation technology (CAT) as shown in Figure 4. EtherCAT also provides for gateways to integrate existing fieldbus components such as CANopen or Profibus. EtherCAT runs under RTX64 in software without the need for any specialized EtherCAT card plugged into the system bus. Running on one or more processor cores, EtherCAT communicates directly with whatever network interface chip (NIC) is used in the system. The individual device can be selected during EtherCAT configuration.

EtherCAT represents an attractive alternative to the often complex and expensive wiring schemes associated with industrial control systems. A single cable can carry multiple control channels along with safety signals with Safety Inspection Level (SIL) 3 certification. In addition, multiple cores can be dedicated to EtherCAT functionality for truly rich control connectivity, all without the expense and power consumption of additional specialized hardware.

IntervalZero's RTX64 has opened the world to Windows-based real-time systems with high-end vision, visualization and rich user interfaces. It has done this by giving the developer the ability to transform the functions of hardware components into software components by harnessing the power of the underlying multicore processing hardware. For the OEM, there is nothing to inventory and the parts can be replicated infinitely. For the software team, there is no need for specialized knowledge of hardware such as DSPs and FPGAs. The code exists in a unified code base and can be managed as such.

RTX64 integrates seamlessly into the Microsoft Visual Studio Integrated Development Environment, and deploys to a single integrated Windows system. It extends Windows, delivering hard real-time precision with bounded latency, and it does so with multicore processors as a scalable natively SMP-enabled solution. Its positive effects on cost, time-to-market, inventory, user experience and raw system performance are revolutionary.