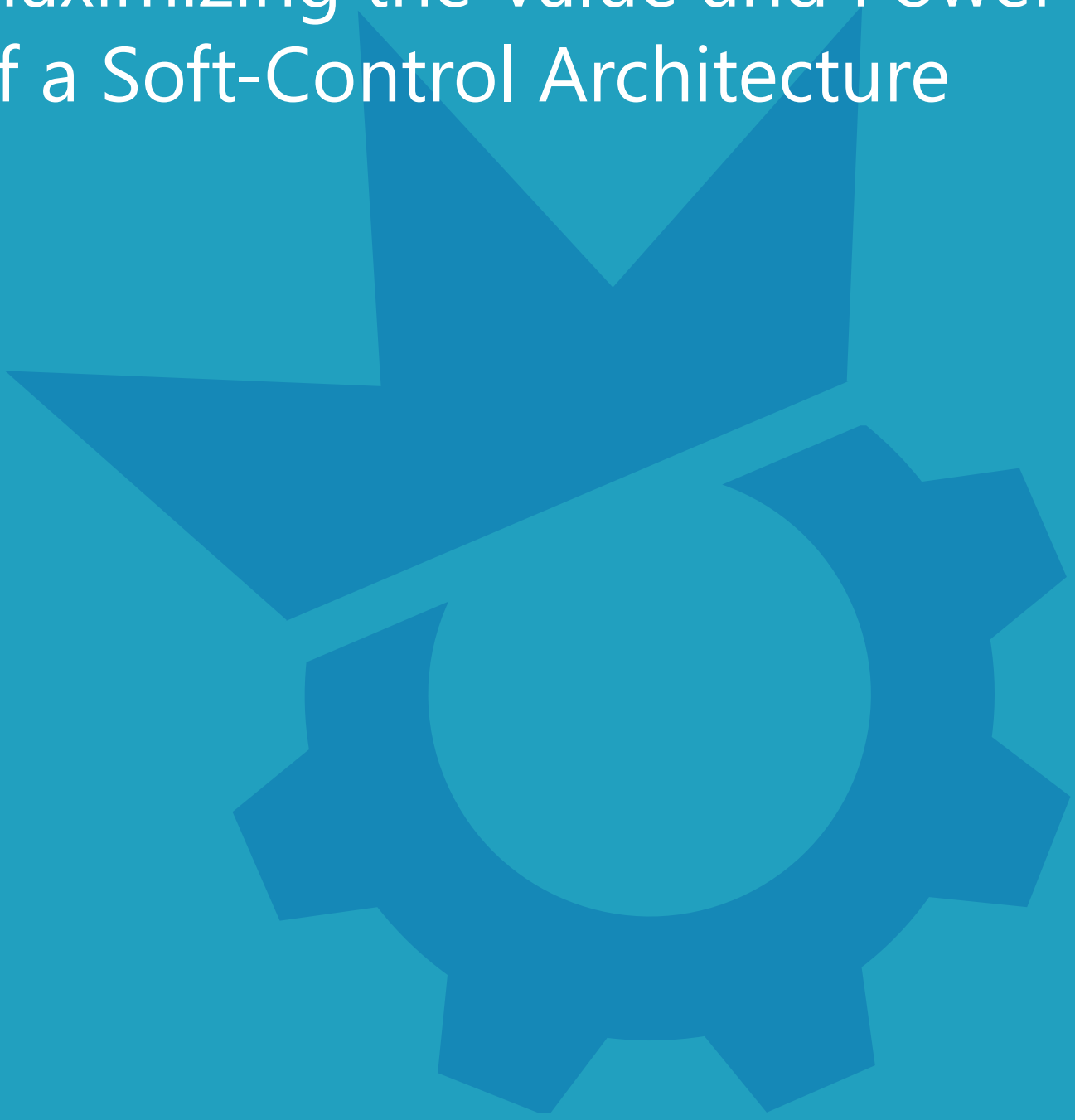
Symmetric Multiprocessing or Virtualization

Maximizing the Value and Power of a Soft-Control Architecture



SYMMETRIC MULTIPROCESSING OR VIRTUALIZATION WHITE PAPER

KINGSTAR

Overview

The continuous advance in multicore technologies is the catalyst for software innovation that is spawning a new generation of embedded systems that are less costly to develop, higher performing, and scalable because tasks run in parallel.

The ideal architecture for these new systems is described in an earlier white paper entitled: "A Soft-Control Architecture: Breakthrough in Hard Real-Time Design for Complex Systems." This white paper presumes a basic understanding of how Soft-Control Architectures can replace FPGA/DSP/PowerPC and proprietary RTOS solutions with a hard-real-time software plug-in for Microsoft Windows to deliver breakthrough cost, performance and scalability benefits.

System developers are already capitalizing on Soft-Control Architectures to gain sustained competitive advantages in markets as diverse as Industrial Automation, Medical Systems, Test & Measurement, and Digital Media.

This white paper examines two prevalent multiprocessing approaches that are competing to deliver on the value of Soft-Control Architecture. They are symmetric multiprocessing (SMP), and asymmetric multiprocessing (ASMP), often referred to as virtualization, or hypervisor technology.

Both approaches have attributes and challenges, but the comparison scale tips heavily in favor of SMP for developers seeking to maximize scalability and minimize latency by fully exploiting the power of x86 multicore – 4 cores, 6 cores, 8 cores, and beyond.

As you will see, virtualization/ASMP does offer short-term gains in terms of initial cost reduction, but is a dead-end technology for a true real-time system that requires scalability, precision and performance – all with minimal latency.

> Virtualization/ASMP is the status quo, albeit on a single chip. This mirror-image status quo is extremely limiting. It lacks flexibility, inhibits performance, and prevents scalability as developers move beyond dual core.

Because SMP-enabled architectures are dynamic – rather than static, as virtualization/ASMP architectures are – they offer developers a wider range of options to streamline and simplify the development processes, while taking full advantage of multicore processing capabilities to deliver systems that change the basis of competition.

This is especially true for developers of embedded systems that have complex Human Machine Interfaces (HMIs) and very demanding hard real-time and control requirements. Large-format medical systems such as a MRIs, and Digital Media mixing consoles, are examples of systems that have these requirements.

As mentioned above, a thorough description of Soft-Control Architectures is available in an earlier white paper, but we are including a synopsis of that paper's highlights to provide context for the comparative information included in this paper.

FPGAs and DSPs have long ruled the market when it comes to hard real-time for motion control and other complex, high-precision and high-performance systems. That is no longer the case. Advancing technologies make it possible for OEMs to deploy Soft-Control Architectures that can displace much of this proprietary hardware.

FPGAs and DSPs have long ruled the market when it comes to hard real-time for motion control and other complex, high precision and high-performance systems. That is no longer the case.

The major trends favoring an SMP-enabled Soft-Control Architecture include:

- Increasingly powerful x86 processor technologies;
- The drive toward more commercial off-the-shelf (COTS) hardware and software;
- Advances in, and the availability of, Ethernet-based field buses;
- Convergence of components in system design; and
- The advent of touch-centered usability – particularly multi-touch – and motion-sensing technologies.

The resulting Soft-Control Architecture that capitalizes on these trends leverages multicore x86. It runs on Windows (including Windows 7 with its touch and gesture technology) on a single multicore chip along with a symmetric multiprocessing-enabled hard-real time plug-in, such as IntervalZero's RTX/RTX64.

Different technologies can be used to build a Soft-Control Architecture, but to deliver the most value, developers must keep in mind 8 key success characteristics:

1. A common Integrated Development Environment (IDE) and world-class graphical user interface (GUI) – Microsoft Visual Studio and the Windows operating system, including Windows 10 and Window Embedded Standard 7;
2. An SMP-enabled real-time subsystem executing directly on multiple assigned processors (not multiple instances);
3. Visibility of the hardware to all real-time processes;
4. The ability to schedule real-time threads across multiple processors, or dedicate certain logic to specific cores, with hooks for load balancing;
5. Direct access to shared data/memory without additional copies and IPC usage;
6. Minimization of hardware requirements – processors, memory, power and footprint;
7. The ability to debug across the cores;
8. The ability to code once and scale automatically.

While SMP delivers on all 8 of these success characteristics, virtualization/ASMP delivers partially on only one – minimization of hardware requirements (#6) – by consolidating what was formerly two-PC systems down to a single PC system.

Virtualization/ASMP does allow many different operating systems/applications to reside on the same multicore hardware, and share peripheral resources such as I/O, serial, Ethernet, USB, etc. For that reason it appears to be a good way to reduce system costs and system complexity.

Applications that formerly ran on separate PC systems can be consolidated onto a single piece of silicon in a single PC system. This cuts the bill of materials costs and allows existing code to be preserved, eliminating the need for rewriting and re-architecting. However, this is where things reach a dead-end. The static nature of virtualization/ASMP architectures prevent them from scaling and therefore from being viable longterm solutions.

For example, with virtualization/ASMP all the elements that were required in a multi-processor configuration – separate real-time operating system (RTOS), separate code base, separate development tools, and separate development teams – are mirrored on a single system. The customer is not managing any fewer instantiations, but rather is adding a hypervisor to manage them.

In short, while there is a gain in terms of hardware costs, there is no accompanying software breakthrough – as there is with SMP – that enables developers to scale up the system on multicore and build systems that differentiate themselves in the marketplace.

For example, as embedded developers seek to leverage more and more cores, by breaking larger functions into different cores for improved performance; or by breaking cores into related functions; or by dedicating cores for specific computational and math functions – all of which can be accomplished with SMP – virtualization/ASMP's mirror-image architectures present significant challenges.

Virtualization/ASMP is the status quo, albeit on a single chip. This mirror-image status quo is extremely limiting. It lacks flexibility, inhibits performance, and prevents scalability as developers move beyond dual core.

A closer look at the other 7 success characteristics shows the comparative strength of SMP and exposes the weaknesses in virtualization/AMSP in delivering a hard real-time Soft-Control Architecture.

1. A common Integrated Development Environment (IDE) and world-class graphical user interface (GUI) – Microsoft Visual Studio and the Windows operating system, including Windows 10 and Window Embedded Standard 7;

While a virtualization/ASMP technology might, or might not, offer a common Development Environment, virtualization does not provide an integrated environment. Each instantiation of the OS must have its own code and is blind to the code in the other silos running in other cores. Virtualization's lack of integration causes management costs to escalate steeply.

2. An SMP-enabled real-time subsystem executing directly on multiple assigned processors (not multiple instances);

> This secondary scheduler provided by an SMP-enabled real-time plug-in such as IntervalZero's RTX/RTX64, has clear visibility across all of the processors. This assures proper utilization and synchronization among the cores.

SMP architectures benefit substantially by having a secondary scheduler – in addition to, and separate from, the Windows scheduler – that is dedicated for the cores on which all of the guest application processing takes place.

This secondary scheduler provided by an SMP-enabled real-time plug-in such as IntervalZero's RTX/RTX64, has clear visibility across all of the processors. This assures proper utilization and synchronization among the cores.

This allows the user to implement a supervisory process that can monitor the activity on the various cores. Based on environment demands or user commands, the scheduler can use the various API calls to distribute or consolidate functions across the cores. This is extremely useful for optimizing throughput, and minimizing power consumption by idling cores.

As an example, let's look at a quad core x86 device where one core is allocated to Windows and the remaining three cores are allocated for the SMP subsystem. Under heavy conditions, all three of the SMP dedicated cores could be fully loaded at 100%, which is an optimal use of the hardware. However, what if the 100% loading only happens at a duty cycle of 20%? This is an inefficient use of the multicore architecture, wasting power and performance. SMP APIs enable users to write a supervising process, or load balancer, that can consolidate the functions onto a single core, in turn allowing the system to idle/park two cores to reduce power consumption.

The same API calls can also be used to distribute the threads/processes to underutilized cores in situations where a single core is overloaded, resulting in greater processing and throughput.

Virtualization/ASMP architectures have a great deal of difficulty implementing a supervisory function or load balancer to help manage the processing among the various cores because there is a separate application image and RTOS for each core. Each application is cognizant only of what is scheduled within that application and also through whatever hardwarebased IPC that has been designed. This means that in order to properly move and schedule threads/processes between cores, the original application would have to have been pre-built with all of the necessary communication and functional support for load balancing, which is not typically in the design specifications for a single core device (i.e. DSP). These load balancing APIs are native to the SMP scheduler.

Additionally, virtualization/ASMP architectures inherit limitations from the legacy multi-processor design. For example, the copied design might have had a number of performance limitations (CPU, memory, I/O, etc) that were present in a DSP that was used in that design.

These limitations needn't necessarily be carried over to a new multicore device, but because the application is literally imported directly over into the virtualization/ ASMP design, the constrictive limitations remain in place. The result: a less than optimal design.

3. Visibility of the hardware to all real-time processes;

With SMP architectures, the guest applications run directly on the cores with unobstructed access to the I/O. In short, there is full visibility of the hardware to all real-time processes. This is not the case with virtualization/ASMP.

With SMP architectures, the guest applications run directly on the cores with unobstructed access to the I/O. In short, there is full visibility of the hardware to all real-time processes.

Because a virtualization/ASMP architecture is simply a direct import of what were once separate applications running on dedicated devices, no consideration is given to how the host and guest applications can fully use the new device's features and capabilities. This lack of coordination ultimately leads to a great deal of hardware contention.

Let's look at the example of a consolidation of one Windows device and two DSPs into a single quad core x86. Each of the different applications had their own exclusive hardware/peripherals that include anything from serial ports, to Ethernet controllers, to USB ports. This is where an virtualization/ASMP architecture requires a hypervisor, or abstraction layer, to manage and arbitrate the peripherals among all of the different applications. This creates the illusion, to the different applications, that they have exclusive rights to a particular piece of hardware. Although functional, this arbitration results in system latency. As the number of applications increase there more contention and the latencies amplify.

Conversely, in an SMP architecture, all system partitioning is done up front regarding which thread/process will own or share a particular piece of hardware (Ethernet, USB, etc). Traditionally in an SMP architecture, the Windows scheduler handles all of the HMI as well as any non-critical processing, while the secondary real-time scheduler handles all time-critical, and CPU-intensive threads/processes.

SMP has the hooks/APIs required to move threads/processes between cores. This is a necessary building block for load balancing and proper usage of a multicore architecture. Load balancing is a key for assuring the best throughput and optimal core utilization.

Lets look again at the example consolidation of one Windows device and two DSPs into a single quad core x86 – this time with an SMP-enabled real-time plug-in to Windows. The first step in re-architecting for SMP is to schedule all time-critical or CPU-intensive threads/ processes to the secondary SMP scheduler while the Windows scheduler retains its functionality. The designer then determines where the peripheral/drivers are to reside: either in the non time-critical Windows subsystem or in the SMP real-time subsystem. Once a driver is assigned, that particular subsystem has unobstructed and exclusive rights to that peripheral. This architecture scales very well because as the core count increases, the SMP scheduler can easily control the threads/ processes and drivers on any of the cores though simple synchronization mechanisms (semaphores, mutexes, etc.).

There are situations where drivers can be shared and in those cases the driver APIs can be exported between the subsystems to share both peripherals and synchronization calls. This illustrates how SMP provides not only a great deal of performance, but also a great deal flexibility on how a system can be built for scalability and parallelism. Scalability and parallelism are becoming increasingly important, given the incredible acceleration of multicore devices from Intel and AMD.

4. The ability to schedule real-time threads across multiple processors, or dedicate certain logic to specific cores, with hooks for load balancing;

SMP has the hooks/APIs required to move threads/ processes between cores. This is a necessary building block for load balancing and proper usage of a multicore architecture.

Load balancing is a key for assuring the best throughput and optimal core utilization.

As discussed above, virtualization/ASMP architectures have difficulty with load balancing because the original system partitions are carried over from the copied legacy design. Threads/ processes are contained within a particular guest application and there is no easy way to move functions between processors without additional copies and considerable complicated coordination among the different cores.

5. Direct access to shared data/memory without additional copies and IPC usage;

Because SMP architectures simply share memory with the host application, it is natural to share data among the different cores/applications. This reduces memory requirements and improves performance by eliminating any coping of data.

Virtualization/ASMP architectures must make multiple copies of the data and the program code because the original design limitations are inherited – for example, the IPC communication and the requirement for exclusive memory for program and data. Buffers of redundant data add latency.

6. Minimization of hardware requirements – processors, memory, power and footprint;

SMP architectures and virtualization/ASMP architectures both consolidate the hardware requirements and reduce the compute hardware costs by 25-50%. However, virtualization/ASMP does not earn full marks because hardware consumption is not optimized and because contention and latency can occur at the hypervisor level that not visible at the application level.

By contrast, SMP architectures have incrementally more efficiencies and smaller footprints because there is a great deal of shared data among the cores as well as between the two subsystems (Windows and SMP). This reduces the amount of memory needed and lowers system cost even further. Also, the greater efficiency and performance enabled by organizing threads/ processes across the different cores often reduces the processor requirements, further lowering system costs.

7. The ability to debug across the cores;

This is very important.

In a virtualization/ASMP architecture, contention and latency can occur at the hypervisor level that is not visible at the application level. This is particularly troublesome – making debugging extremely difficult, and system failure a major concern.

For a tool set to be able to debug a multicore system properly, it has to be aware of all of the operating systems and have visibility across the entire system. In a virtualization/ASMP architecture, because the original applications and RTOSes are being imported in directly, the IDE is not cognizant of the guest applications running on the other cores.

In this case, separate tool sets are required to debug between the different cores or applications. This is not particularly challenging in a dual core system, but becomes increasingly overwhelming as the system grows to 4 cores and beyond.

Imagine the complexity in debugging 4-core or 6-core system where you have 4 or 6 more separateIDEs trying to debug the different cores. The individual IDEs/debuggers, although powerful in their own right, have very limited visibility into the other environments. This limited debug visibility issue is compounded when dealing with the added latency a hypervisor introduces as it arbitrates shared peripherals that were once exclusive to the different applications.

It is also worth noting that when dealing with time critical applications, the added latencies created by a hypervisor can present an unknown time constant that has to be taken into account at the application layer.

Finally, in addition to the debugging complexities, virtualization/ASMP architectures usually require the user to maintain all the different tool sets and operating systems. As the core count increases, the maintenance of these different tool sets adds cost and creates a support burden.

> In a virtualization/ASMP architecture, contention and latency can occur at the hypervisor level that is not visible at the application level. This is particularly troublesome – making debugging extremely difficult, and system failure a major concern.

SMP architectures have a much simplified approach to development tools and debugging.

SMP applications are built within the Visual Studio IDE and it is fully aware of both the Windows and SMP schedulers. The SMP subsystem uses a secondary scheduler that presides over all of the cores in the real-time subsystem. As a result of having a single scheduler for the SMP subsystem, there is no added debugging complexity as the number of cores/applications increase. The RTX debugger for Visual Studio has full control and visibility across all of the cores as well as the two schedulers – Windows and SMP.

Applications within a SMP architecture have direct interaction with and ownership of their peripherals, eliminating any extra latencies that would have to be accounted by the applications. This tight integration makes development and debugging straightforward as the user can easily set breakpoints between Windows and the SMP subsystem all within Visual Studio. SMP's use of a single IDE simplifies and streamlines debugging and tool maintenance extremely and is very cost effective.

As a result of having a single scheduler for the SMP subsystem, there is no added debugging complexity as the number of cores/ applications increase.

8. The ability to code once and scale automatically.

This was touched upon above, but because parallelism is so important, this aspect deserves a fuller explanation.

SMP architectures include APIs to assign processor affinities and ideal processors. These APIs allow developers to code once for the ideal multicore architecture, providing the ability for the software to automatically scale to devices as more cores become available.

On the other hand, after the first transition from separate processors to a single piece of silicon, it becomes difficult in a virtualization/ASMP environment to properly utilize the different core multiples when designing for a variety of systems.

Without a major rewrite of the code, the virtualization/ASMP design must put multiple copies/images of the original guest application on each of the additional cores. This is redundant and wasteful because there is no ability to improve performance and throughput by grouping threads/processes that would benefit by running on particular cores.

Developers are right to be very concerned that virtualization/ASMP does not mitigate the complexity and inefficiency that existed in the traditional separate-processor systems – multiple tool environments, multiple development teams and multiple operating systems.

And in consolidating from separate chips to a multicore architecture in which the peripherals are shared, resource contention can become a significant challenge for the various development teams.

Sharing peripherals that were formerly dedicated to each application inevitably means that some operations have to wait for access while a needed peripheral is busy. This is not only inefficient, but antithetical to the goal of getting the most processing speed and power out of the multicore hardware. In short, idle silicon is a waste of resources. For developers to truly reduce system costs they must maximize all processing potential all the time.

As detailed earlier, these challenges simply do not exist in SMP architectures.

In short, although virtualization/ASMP may be a suitable approach for single-task or less complex environments – reasons why desktop virtualization is readily embraced – it does not measure up as a solution for complex multi-task systems where hard real-time, precision, flexibility, time-to-market acceleration and scalability are mandatory.

In fact, for virtualization/ASMP to be a credible option for complex embedded systems, a Windows hypervisor extension would be required. This would allow a realtime SMP product to plug in and extend the core and peripheral access.

With SMP, developers have many options in rethinking and redesigning their systems, not only to get the most out of multicore capabilities, but also to enhance the development processes for long-term economic benefits and scalability.

By augmenting the Windows/x86 system with an SMP-enabled, hard real-time subsystem that adds a second independent scheduler, systems developers get a Soft-Control Architecture that is well suited for maximizing multiprocessing capabilities, as well as for delivering significant economic and performance benefits that extend far beyond system consolidation.

Conclusion

At best, virtualization/ASMP represents a short-term, low-value consolidation path from multiple systems to multicore. However, for a truly simplified and streamlined architecture that is high-performing, scalable, efficient. and built for long-term value, an SMP-enabled Soft-Control Architecture is recommended.

Symmetric Multiprocessing (SMP) vs. Virtualization

8 Key Design Elements for a Windows/X86 Real-Time Embedded Solution	Benefit	SMP	Virtualization ASMP
1 A common Integrated Development Environment (IDE) and world-class graphical user interface (GUI) – Microsoft Visual Studio and the Windows operating system, including Windows 10 and Window Embedded Standard 7	Consolidation & Productivity	●	○
2 An SMP-enabled real-time subsystem executing directly on multiple assigned processors (not multiple instances)	Performance & Maintainability	●	○
3 Visibility of the hardware to all real-time processes	Control	●	○
4 The ability to schedule real-time threads across multiple processors, or dedicate certain logic to specific cores, with hooks for load balancing	Control & Performance	●	○
5 Direct access to shared data/memory without additional copies and IPC usage.	Quality & Performance	●	○
6 Minimization of hardware requirements – processors, memory, power and footprint	Performance	●	◐
7 The ability to debug across the cores	Performance	●	○
8 The ability to code once and scale automatically	Scalability	●	○