

## Meeting the Challenges of Connected Device Design Through Real-Time Virtualization™

We are surrounded by connected devices such as mobile phones, digital cameras, and hand-held games, everyday. In a fast-paced, technology-driven society, we continuously expect our connected devices to have more and more functionality with the ability to use them from anywhere, at any time. Moreover, these connected devices must perform faster, possess higher bandwidth, consume less power and operate with increased autonomy. The resulting increased product complexity requires complex solutions, especially at the software level. Minimizing the complexity of the solutions as much as possible enables new connected devices to be delivered faster and at a lower cost.

This whitepaper describes some of the technical and market challenges faced by connected device designers and explains how a real-time virtualization solution simplifies many of these challenges. It discusses how other solutions such as adding hardware and migrating legacy application software are used to satisfy connected device requirements but are determined not to be sufficient when compared to using real-time virtualization technology.



## Main Challenges Faced by Network Equipment Designers

*Mobile phones have to complete the full design cycle in less than one year.*

*Cameras and set top boxes have similar cycles.*

*New generations of network equipment must deploy ever-growing volumes of legacy software, on hardware with higher density and faster processing capabilities.*

*The primary usage of any phone is to enable its user to call or receive a call from someone else.*

*Similarly, the primary role of a set-top box is to decode images and display them on a TV screen.*

### **Devices must be released with an ever-decreasing Time To Market window.**

This is a well-known issue: each new generation of devices has to be designed, developed, tested, and delivered in an ever-decreasing time frame. These Time to Market constraints have important consequences for both hardware and software design choices. There are fewer opportunities to design, develop, and test new implementations of existing services, even though the final result may simplify the design of future product evolutions. Consequently, new products must re-use validated existing software as much as possible.

### **Reducing the Total Cost of Ownership (TCO) of the devices is critical to competitiveness.**

Some embedded systems were designed and built quite a long time ago – 15 to 20 year-old designs are not unusual. This is especially true for software used in telecommunications network equipment. New generations of products are based on newer hardware and incorporate faster processing capabilities, multi-core or symmetric multiprocessing (SMP). This reduces the TCO because a single new-generation board now performs the work of several older boards. Adapting the previous generation's software to such new hardware is a challenge that requires a lot of effort and time.

### **To provide their primary service, while keeping Time To Market short, devices must leverage legacy software.**

Embedded device functionality has long been provided by dedicated hardware and software. Such dedicated software, mostly autonomous and isolated from the outside world, is often highly specialized and has undergone comprehensive test, validation, and certification processes. The ability to leverage such legacy software amortizes the cost of previous development and reduces the cost of new products. This is especially true for dedicated protocol and modem stacks used in wireless equipment, such as mobile phones and network infrastructure.

**To be attractive, devices must be increasingly connected and provide end-users with broad and rich functionality.**

*Mobile phones provide Internet access, and will use higher and higher bandwidth.*

*Set top boxes no longer display only images on a TV set but are now connected to high-speed networks, using IP protocol stacks.*

*Reusing an unmodified legacy mobile phone radio stack in its original runtime environment eliminates lengthy debug and validation.*

*The user is expecting to access traditional desktop native applications – such as mailers, browsers, games, audio and video players – on mobile phones and set top boxes.*

The deployment of ever-more powerful processors and more advanced hardware, combined with the availability of high-speed communication networks enables prolific device connectivity, most commonly via IP based networks. Although many RTOS' include IP stacks, new features such as encryption, tunnelling and virtual networks are required in order for the connected devices to fully deliver their services. IP protocols are first developed for feature-rich OS' (also known as RichOS) such as Linux and Windows. Adapting such complex software to run in a different operating system is both costly and time consuming. It is also difficult and expensive to keep pace with the frequent development of new extensions, such as VLAN, IPsec, VPN, etc.

A protocol stack must be fully debugged and validated to ensure that it operates with any implementation of the same protocol. Such debug and validation requires a great deal of effort (and budget!), so re-using such a stack – with no modification – in a new device saves time and money. In embedded devices that must simultaneously execute dedicated real-time applications, communication stack and an IP protocol stack, re-use of the OS on which each stack has been previously implemented is often an attractive option.

The increasing processing and networking capabilities of connected devices enable more powerful applications to be offered to the end-user without compromising the performance and other features of the device. Again, rather than developing such applications from scratch, it is often more convenient, end-user friendly, and faster to rely on existing applications originally developed for a RichOS, such as email clients, web browsers, and multimedia players. This is much more efficient than porting these applications to the RTOS which runs time-critical services and stacks.

***Dilemma faced by connected device designers:***

*“How to combine multiple applications stacks – including OS, protocol stacks, and the applications themselves – originating from different operating environments without modification?”*

## An ideal design should enable both legacy and new software to run simultaneously with the lowest Bill of Materials (BoM).

*Why design a mobile phone with a baseband and an application processor if the latter is powerful enough to hold both loads?*

### Simple hardware designs result in lower costs and shorter Time To Market.

An ideal solution would enable a system designer to keep hardware design as simple as possible. One approach is to design the device by increasing the number of processors, one of which runs legacy or real-time software on its own (real-time) operating system, and another that runs open or third-party software on a high-level operating system such as Linux or Windows CE. This option increases the complexity and cost of the design, and significantly raises the BoM. In addition to the added processor costs, memory resources are duplicated, unless expensive dual-ported memory is used. Input/Output (I/O) peripherals must also be duplicated, or more complex access must be provided. Of course, all of this extra hardware increases power consumption.

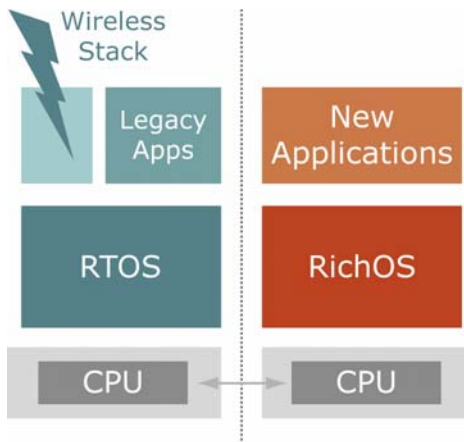
### A simple software-only design is preferable.

An ideal solution would be able to run existing protocol and application stacks originating from different software environments, without modification. In order to ensure the device's operation, these somewhat contrary goals must be managed while preserving the respective features of the applications and their quality of service (QoS) – real-time guarantees must be maintained by the new design. Since software from two or more environments have to run simultaneously, they must not compromise each other and be mutually isolated in order to deliver their intended behavior with the requisite performance. Nonetheless, isolated applications from the two environments need to cooperate through communications and be able to concurrently access common I/O peripherals.

We will now examine the main potential implementation approaches and compare them with the ideal solution requirements discussed above:

- Hardware extensions
- Software porting and/or integration
- Real-Time Virtualization.

## Alternative Implementation Choices



### Hardware Extensions

The software required by new generations of connected devices is a combination of legacy real-time environments and standard, open source and/or third-party software. One way to consolidate these two environments is to provide each with its own hardware. Each environment then runs in parallel on its own dedicated processor. This approach requires little or no software modification; guarantees that applications in one environment are isolated from applications in the other; ensures that real-time applications run on their own real-time OS, and all receive the resources they need to operate correctly. Traditionally, these two environments would communicate through appropriate software layers stacked on top of a physical link – such as a bus or memory – that connects the two environments.

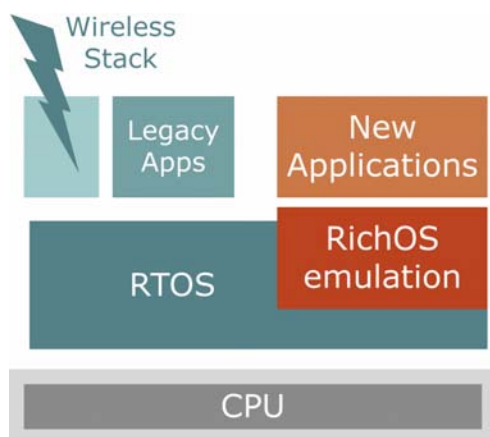
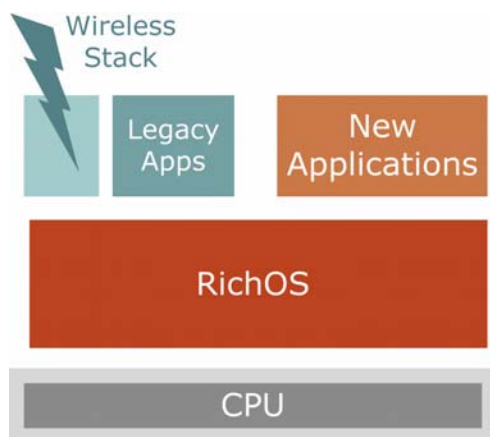
Such an approach seems attractive because:

- It requires “only” an additional processor, some extra memory and possibly a few extra I/O devices. The resulting design is somewhat modular in that products with fewer features may be derived from current ones, without the additional processor.
- Software integration and validation are straightforward because the environments operate separately.
- Only cooperation and interactions between the two environments require some attention. Software can be developed at low cost and in a timely fashion.

However, this approach has some major drawbacks:

- It requires a more complex board design. Time saved on the software development cycle may be lost in the hardware design.
- An additional processor on a board implies a more expensive BoM. Device unit cost is also impacted by the requirement to deploy additional memory or perhaps some dual-ported memory, and various devices may need to be duplicated since accessing them from two processors may be too complex.
- More processing power, devices and memory implies a bigger and heavier device, occupying more space and more importantly consuming more power, reducing the “time between charges” of battery powered devices.

*Some mobile phones currently have two audio devices: one dedicated to the baseband processor for voice and one dedicated to the application processor for music!*



## Software Porting And/Or Integration

This approach involves merging or porting the software environments to run together on the same processor with a single OS environment.

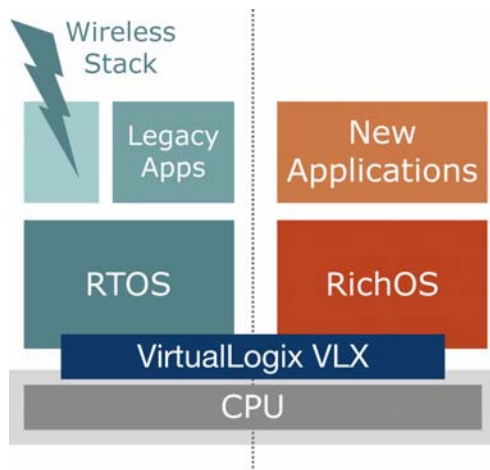
Most applications use well-defined and quite portable interfaces to inter-operate with the OS. Therefore, applications may be ported from the legacy embedded RTOS to the RichOS environment, or the other way around. Emulation libraries may be created to enable applications from one OS to run on another OS.

The advantages of this approach are:

- Hardware design is much simpler than that of the hardware extensions approach previously described. Hence, hardware development time is shorter and the BoM is lower.
- Integrating all software provides the device with legacy applications and protocol stacks as well as with open or third-party applications and IP protocol stacks. Once integrated, all software runs in a single environment, avoiding the burden of dual environments. Moreover, if the selected environment is a RichOS, tracking its evolutions (IP protocol stack in particular) is much easier.

Unfortunately, there are many drawbacks to this software integration approach:

- Merging different software environments requires a lot of redesign, porting, and validation effort. This increases development costs and Time To Market.
- Legacy OS environments are often well adapted to provide real-time guarantees, while RichOS are usually not. The use of emulation libraries may impose timing and performance penalties. RichOS are well adapted for rich file systems or user interface support, but legacy OS are much more restrictive on this point. Hence, no single environment of this kind can simultaneously provide the guarantees and QoS required by both sets of applications.
- In addition, such an integrated software stack should be configurable so that scaled down versions may be run on smaller hardware configurations where RichOS applications and services may not be necessary.



## Real-Time Virtualization

The need to run multiple software environments on a single-processor based architecture is not unique to new generations of embedded systems. IT departments have been using a proven working solution to such an issue: hardware virtualization.

However, the heavy weight virtualization techniques used on IT servers do not apply directly to connected devices, which have smaller memory configurations and require higher performance, especially for real-time applications.

In addition, processors used in connected devices are generally very different from those used in IT servers, limiting the re-use of IT virtualization solutions in the embedded space. In order to achieve the appropriate level of performance – in particular interrupt latency – the overhead imposed by typical virtualization technologies is a roadblock. There is a need for a virtualization solution that uses a different approach to achieve the requisite performance level.

Paravirtualization adapts the OS to the virtualization engine. It affords an optimal trade-off that achieves the requisite performance, provided that the adaptations are (a) minimal, (b) well isolated in the low layers of the OS, and (c) remain fully invisible to applications, drivers and protocol stacks.

To fulfill various performance, security, and feature set requirements, the virtualization technology must be modular and configurable to easily achieve the trade-offs required by the final product.

To ensure secure, efficient and timely device operation, hardware resources must be managed on a “fine grain” basis. That is, it must be possible to allocate memory to any of the environments as needed, as well as to provide flexible scheduling of activities, regardless of which operating environment is used, while simultaneously preserving real-time guarantees. Moreover, it must be possible to exclusively allocate an I/O device to one environment, or to share it between several environments.

In addition, communication services – such as IP protocols, modem AT commands, messaging – between applications running in different environments must be configurable.

## VirtualLogix VLX Virtualization Technology

VirtualLogix VLX virtualization technology enables multiple operating systems, referred to as “guest OS”, to run simultaneously on the same processor while preserving real-time characteristics. Guest OS's run independently of each other, but can cooperate by means of efficient inter-OS communication mechanisms.

A thin abstraction layer, VirtualLogix VLX, manages key system resources to isolate the guest OS's from the underlying hardware. More precisely, VirtualLogix VLX virtualization technology relies primarily on the partitioning of resources between guest OS's and on the virtualization of resources that cannot be partitioned.

*VirtualLogix VLX manages key resources, through partitioning and virtualization, isolating guest OS's from hardware.*

Typically, physical memory is partitioned between guest OS's, while the CPU, FPU, MMU or other system components, such as the real-time clock and interrupt controller, are virtualized by VirtualLogix VLX.

This approach enables the VirtualLogix VLX virtualization technology to be applied to embedded and real-time systems.

### Partitioning

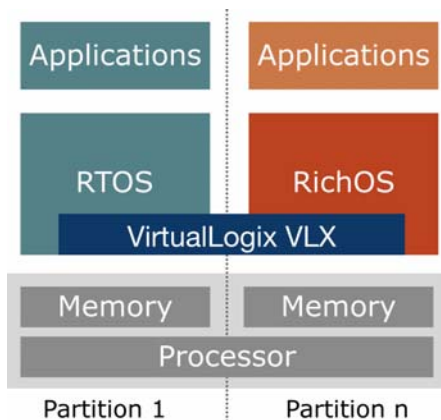
Physical memory is partitioned, and each partition is allocated to a given guest OS. Thus, each guest OS may use its own native memory management mechanisms and policies, without interfering with other guest OS's. This approach allows OS's that make use of the MMU, e.g. Linux and Windows CE, and those that do not, e.g. Nucleus and VRTx, to run cooperatively on a single processor.

I/O devices used by a single guest OS are assigned exclusively to that guest OS. Native device drivers can thus be re-used with no modification.

### Virtualization

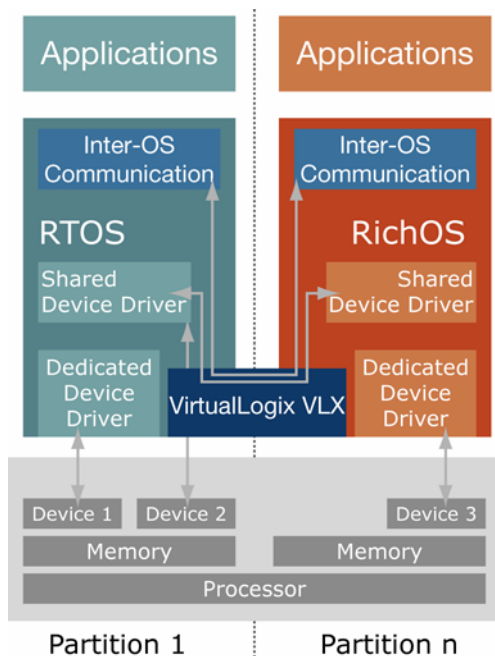
Hardware resources required by several guest OS's, such as the CPU and real-time clock, are virtualized so that they can be shared between all guest OS' that need access to them.

For efficiency, performance and footprint reasons, VirtualLogix VLX relies on paravirtualization techniques, meaning that the guest OS kernel is adapted to the virtualized platform. These changes are comparable in both effort and scope to porting that same OS to a variant of the board based on a similar underlying CPU and device hardware. Supporting a new guest OS's is therefore straightforward.



VirtualLogix VLX virtualizes the CPU, FPU and MMU, if any. The CPU is shared among the guest OS' based on scheduling policies that guarantee that the real-time guest OS (and its applications) is assigned the highest priority and can thus maintain its real-time guarantees. Once a guest OS has been granted CPU access, the OS uses its native policies to schedule its own applications and services.

If present, the MMU is virtualized so that each guest OS may use it for its own purposes. As discussed above, usage of the MMU by one guest OS is independent of the usage of the MMU by another guest OS.



## Device Virtualization

Running different OS's simultaneously on the same processor is only one part of the problem. The guest OS's also need to communicate and share devices or other hardware resources.

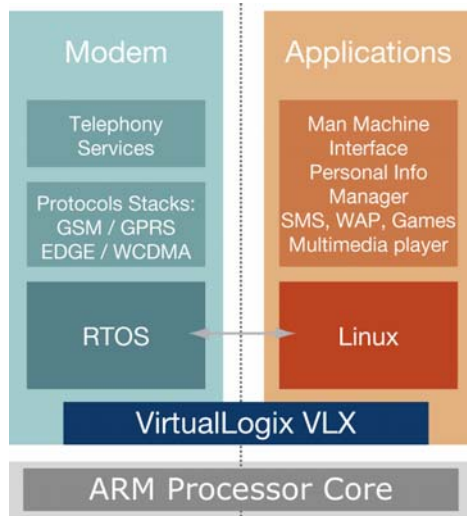
A typical OS supports multiple processes, provides them with memory allocation and scheduling policies, and offers services such as synchronization, shared access to system and network interfaces and inter-process communication.

Similarly, VirtualLogix VLX virtualization technology provides each guest OS with synchronization (a cross-interrupt mechanism), shared access to devices such as disk controllers, network interfaces, serial lines and inter-OS communication mechanisms, through virtual devices (virtual Ethernet or virtual UART).

I/O devices fall in one of the three following categories:

- Dedicated I/O devices are used by only one guest OS, and require no driver modification
- Shared I/O devices may be used by more than one OS; they require specific cooperating drivers to permit physical I/O device sharing between various guest OS'. VirtualLogix has developed a portfolio of such drivers, such as Ethernet, audio, frame buffer, disks, UART, modem, keyboard, etc.
- Virtual devices do not correspond to a physical device, but are used for inter-OS communications. Again, VirtualLogix has several such virtual devices available, such as UART, Ethernet, etc.

### Use Case: Mobile Phone



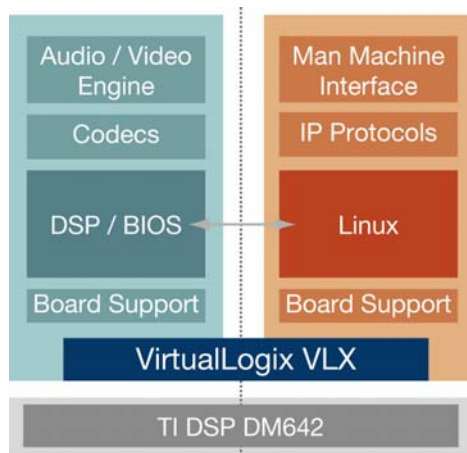
The combination of highly integrated chipset solutions with Linux mandates new software architectures and technical choices that need to be validated against crucial features such as real-time support, robustness and efficiency.

VirtualLogix VLX for Mobile Handsets enables new mobile phone integrated architectures to run both modem and additional applications on the same processor core by sharing hardware resources. With VirtualLogix VLX, the existing modem protocol stack, validated on its own RTOS, can be re-used without modification, while Linux with its drivers, IP protocol stacks and applications can be introduced at a minimal cost.

VirtualLogix VLX high-performance virtualization solution preserves RTOS native behavior while providing the advanced features needed for mobile phones, such as fine grain scheduling, power management, and optimized cross-OS communication paths.

VirtualLogix VLX enables Philips Semiconductors to re-use its mobile phone stack, saving man-years of engineering effort and releasing the product to the market faster.

### Use Case: Set Top Box



VirtualLogix VLX for Digital Multimedia enables a DSP operating system and a Linux system to run simultaneously on a single DSP. This solution is particularly attractive for low cost IP set-top boxes, and has been implemented with the Texas Instruments (TI) DSP/BIOS operating system running on the TI DM642 digital media processor.

Meeting the goal of simultaneously decoding incoming streams and running Linux applications typically calls for two distinct processors with different requirements to support each operating system. However, the VirtualLogix VLX partitioning and virtualization solution enables Linux to run side-by-side with DSP/BIOS real-time operating system on a single TI DM642 DSP. This unique combination enables the re-use of Linux applications while executing the codec efficiently on DSP/BIOS.

For example, VirtualLogix VLX enabled Amino to save approximately 20-25 percent on the BoM by eliminating the need for an additional processor and its associated hardware components.

## A Unique and Mature Solution

### Examples:

*AmiNET124 set top box  
Alcatel 5020 MGC softswitch*

### Examples:

*ARM 926  
TI C6000 DSP  
Intel Pentium*

### Examples:

- *VxWorks*
- *Nucleus*
- *DSP/BIOS*
- *Linux 2.4, 2.6*
- *Windows CE*

### Already deployed in end-user products

VirtualLogix VLX technology is used in products shipping and deployed today. Other products based on VirtualLogix VLX will reach the market very soon.

### Runs on many diverse hardware platforms

VirtualLogix VLX supports key processor families including ARM, TI DSP, Intel x86, PPC and platforms, with or without a MMU. Multi-processor platforms are also supported. This wide variety of supported hardware demonstrates the maturity of VirtualLogix VLX technology and its ability to easily fit most platforms.

### Supports many diverse OS

VirtualLogix VLX supports many different commercial and proprietary RTOS. Today, more than 7 different RTOS have been adapted to VirtualLogix VLX. It also supports a variety of RichOS such as Linux and Windows CE. This demonstrates the ease of adaptation to VirtualLogix VLX and the maturity of its technology.

### Offers flexibility and modularity

In addition, the VirtualLogix VLX architecture has been designed for flexibility. VirtualLogix VLX components can be enhanced with modules to provide additional services, such as increased security, platform management, or compatibility, depending upon the needs of the target product.

### Comprehensive I/O peripheral device management

Most connected devices must manage a wide range of I/O peripherals. Some, such as modems, must be handled by the RTOS. Others are better managed by the RichOS. However, many I/O peripherals must be accessed by both environments – frame buffer and audio device are examples in a mobile phone. To handle such constraints, VirtualLogix VLX provides a comprehensive set of drivers, enabling the guest OS's to transparently cooperate while accessing such I/O peripherals.

Communication mechanisms are not hardwired by VirtualLogix VLX. They can be simply configured as virtual device drivers and made available to applications through the most appropriate interfaces. This flexibility ensures the best possible throughput between applications running on various guest OS's.

## Performance

The VirtualLogix VLX architecture introduces almost no overhead to the RTOS. A negligible and bounded overhead occurs only when an interrupt occurs while the second guest OS is active, enabling real-time applications to run undisturbed.

The performance of the RichOS is highly dependent upon the load of the RTOS. However, analysis shows that the overhead imposed on the RichOS is minimal when the RTOS is inactive, and increases only slightly with the activity of the RTOS.

## Advanced Features

VirtualLogix VLX is a deployed solution that has already solved many of our customers' complex product design challenges.

Running several OS's, side-by-side might raise some concerns regarding the global scheduling of tasks, independently of whether these tasks run within one guest OS or another. VirtualLogix VLX provides several scheduling policies with flexible control of task scheduling. As an example, avoiding CPU starvation of a guest OS is a guarantee against some Denial of Service (DoS) attacks. Another policy enables fine grain scheduling between activities running in different guest OS's.

Security within connected devices is an increasingly important issue. The ability to run isolated environments with different security levels on the same device permits to secure sensitive end-user data while offering the possibility to download games and applications without compromising device integrity. VirtualLogix VLX is a sound foundation for such configurations.

## Summary

VirtualLogix VLX virtualization technology solves many of the problems faced by connected device designers and enables them to:

- Keep the hardware design simple, lowering the BOM, development costs and improving Time To Market,
- Port different software environments to a single processor based hardware.
- Run legacy, real-time and RichOS software without complex and lengthy redesign and integration.
- Reuse legacy real-time applications without modification and with guaranteed response times.
- Optimize the system to deliver the requisite product features and performance

### VirtualLogix

292 Gibraltar Drive, Building 104  
Sunnyvale, CA 94089, USA  
Tel +1 408 636 2800  
Fax +1 408 636 2815