WP474 (v1.0) March 31, 2016

Enabling Virtualization with Xen Hypervisor on Zynq UltraScale+ MPSoCs

Xen hypervisor on Zynq® UltraScale+™ MPSoC provides robust hardware-accelerated virtualization and ease of use, helping embedded system designers get the most out of their hardware investment.

ABSTRACT

Virtualization is a staple for desktop systems but has long been a complex proposition for embedded systems designers, who need to optimize SoC system utilization and performance.

Traditionally, the pain of virtualization in the embedded space was caused by the lack of appropriate hardware resources to make the solution easy to implement while providing satisfactory performance. Thus, systems that required heterogeneous software stacks running on the same processor had either to resort to manual management of the various software stacks or to accept the increased latency and decreased performance characteristic of unaccelerated virtualization.

The ARM®v8 architecture at the heart of every Zynq UltraScale+ MPSoC enables true hardware- accelerated virtualization to alleviate these implementation roadblocks. In addition, Xen hypervisor provides an ease of use that sets the stage for desktop-class performance and productivity in embedded systems. Xen hypervisor running on a Zynq UltraScale+ MPSoC provides a system designer with a complete solution, unleashing the full potential of the embedded system design.

Introduction

Virtualization has conquered the desktop by allowing multiple software stacks to run simultaneously on the same processor, and embedded systems are now the next frontier! Popular packages such as VMWare and VirtualBox have made the practice of virtualization commonplace for desktop users. This type of software is seen as a productivity enhancer for desktop computers, but the same principles can also be utilized to get the most out of an embedded System-on-a-Chip (SoC) such as the Xilinx Zynq UltraScale+ MPSoC.

The role of virtualization varies from system to system. For some designers, virtualization allows the processor to be kept fully loaded at all times, thus saving power and maximizing performance. For others systems, virtualization provides the means to partition the various software stacks for isolation or redundancy.

Traditionally, the pain of virtualization in the embedded space was caused by the lack of appropriate hardware resources to make the solution easy to implement while providing satisfactory performance. Thus, systems that required heterogeneous software stacks running on the same processor had either to resort to manual management of the various software stacks, or accept the increased latency and decreased performance characteristic of unaccelerated virtualization.

The ARMv8 architecture at the heart of every Zynq UltraScale+ MPSoC enables true hardware-accelerated virtualization to alleviate these implementation roadblocks. In addition, Xen hypervisor provides an ease of use that sets the stage for desktop-class performance and productivity in embedded systems. Xen hypervisor running on a Zynq UltraScale+ MPSoC provides a system designer with a complete solution, unleashing the full potential of the embedded system design.

Why Virtualize Embedded Systems?

The decision to virtualize is usually driven by one of these three system design characteristics:

- The processor must remain as fully loaded as possible for performance specifications.
- Software isolation or partitioning is needed for safety, scalability, and/or reliability.
- Scaling or redundancy is needed to meet reliability requirements.

Hardware-accelerated virtualization lies at the heart of the Zynq UltraScale+ MPSoC's ARMv8 architecture. Not only does this synergistic architecture make each of these scenarios possible; it actually allows them to be implemented with *minimal effort*. It does this by providing each guest software stack with an isolated sandbox. Without this hardware acceleration, these systems become much more complicated, a real challenge to implement and manage.

Optimized System Loading

Careful management of system loading is a common challenge for embedded systems, but is also one that presents a number of significant challenges when operating without the benefit of a hypervisor. Traditional operating systems such as Linux are very capable of handling symmetric multiprocessing (SMP) tasks where all of the processor cores are under its control. But, what if there is not a task for every processor? Linux idles these processors when they are not in use. Multiple instances of Linux can be virtualized as-is (e.g., no modifications to Linux), each running its own set of tasks. New instances of Linux can be brought online as needed. Conversely, if demand is low, those instances of Linux can be shut down. Consequently, the processor can be spun up or idled as needed but the system at-large can be kept busy nearly continuously. In addition, similar strategies can be employed to manage multiple dissimilar software stacks. There is no requirement for all of the virtualized operating systems to be the same, adding flexibility and modularity to the end system.

Optimized Software Isolation and Partitioning

Software isolation and partitioning is another common use case that can present significant challenges in an unmanaged environment. It is similar to the system loading scenario, but includes the additional requirement that each software stack cannot interfere with any of the others running simultaneously. The simplest example of this scenario is two real-time operating system (RTOS) stacks running side-by-side. Traditional methods of handling these demands involve additional software complexity to ensure that each RTOS interacts with only the finite resources it is allocated. By comparison, the sandbox paradigm employed in a virtualized system allows each RTOS to have full control over the resources available in its sandbox. It treats this sandbox like a complete system and utilizes it without knowing about any other software that might be running on the system. This sandbox paradigm greatly reduces the amount of code that is tied to a specific hardware platform, making the code much more portable. In addition, the RTOS never needs to be aware of any other software in the system, which makes it much less complex. This freedom from dependencies can be a tremendous boon for developers who want to write their code only once and have it deployed on many different systems.

Optimized Scaling and Redundancy

For embedded systems, scaling and redundancy demands grow as SoCs increase in performance and capability.

Scaling requires that the amount of software loaded on the processor increases along with the system demands. For example, a high-performance compute environment requires that additional instances of a Linux-based operating system be brought online to service additional requests from users. By leveraging virtualization, identical copies of the Linux system can be brought online as needed. Later, these instances of Linux can be shut down as the demands on the system diminish.

Redundancy requires that particular services remain available, even as the system goes through times of stress. For example, an RTOS can provide critical monitoring of particular system functions. What if the RTOS were to fail for some reason? Using virtualization, a system monitor can detect a failure and can then either restart the RTOS or start up a new instance of it to minimize or eliminate downtime for those critical system services.

Why Choose Xen

When choosing a hypervisor solution, it is important for it to be robust and reliable. In addition, it must have active development to keep pace with changes happening in the world around it. Xen hypervisor is just such a solution.

Xen started as part of the larger *Xenoserver* project at Cambridge University in the late 1990s. It was released into the open source community in the early 2000s and came under the umbrella of the Linux Foundation in 2013. Under the auspices of the Linux Foundation, Xen has become the *de facto* hypervisor solution for Linux-based operating systems.

While Xen's traditional architecture has been x86-compatible, recent hosts development has made it a robust solution on ARM architectures as well. Xen takes full advantage of ARMv8's underlying virtualization hardware, including the System Memory Management Unit (SMMU).

Xen is provided free of charge with a standard GPLv2 license and has an active user community that develops new features and is an extensive resource for technical support. For those integrators requiring commercial servicing and support, vendors such as DornerWorks provide professional support and structures for Xen.

Software support is also a key differentiator when choosing a hypervisor solution. Xen itself is a Type 1 hypervisor, meaning that it runs directly on the underlying hardware rather than on a host operating system like Type 2 hypervisors VMWare or VirtualBox.

Xen hypervisor divides guest operating systems into *domains*. It uses a special management interface, *Dom0*, to control the runtime operation of the hypervisor. This domain provides specialized software infrastructure for the Xen hypervisor. This operating structure is mandatory for Xen to work properly. Dom0 utilizes privileged software within its kernel as well as special management drivers that can access the underlying hardware directly. One of the most popular Dom0 operating systems is Linux, which has robust support for Zynq UltraScale+ MPSoCs.

All standard guest operating systems exist in unprivileged domains, referred to collectively as DomU. A variety of guest software, including high-level operating systems like Linux, real-time operating systems like FreeRTOS, and even bare-metal code, are natively supported by Xen for use in DomU. Any combination of host and guest is entirely driven by the needs of the system designer. Comparatively, most commercial hypervisor solutions support only a limited number of guests, and in very specific configurations—typically, those developed by the hypervisor providers themselves.

Integrating Xen Hypervisor with Zynq UltraScale+ MPSoC Is Easy

Even given the best possible technological components, a hypervisor solution is likely to see little use if it is not straightforward to implement in user systems. Xen benefits greatly in this regard by being so closely related to the Linux kernel. Enabling Xen hypervisor support is no different than enabling any other feature in the Linux kernel. In addition, management of DomU sandboxes can be handled either manually or via a suite of tools called the Xen Tools.

For designers using Linux as Dom0, installing the Xen Tools can be done simply and easily at the command line. The Xen Tools can be built and installed from source code or via common package managers such as RPM or APT.

Once installed, Xen Tools can create, manage, and destroy DomU environments from within the Linux user space with a single tool called *xl*.

Creating a new virtual machine is as simple as creating a plain ASCII configuration file describing the virtualized environment. It specifies details such as:

- The amount of memory to be allocated for the virtual machine
- The number of virtualized CPUs
- Networking details
- Disk image files

The configuration files can also be easily managed with text parsing tools and revision control software. Refer to the example code below:

```
# This configures an HVM rather than PV guest
builder = "hvm"

# Guest name
name = "My Virtual Machine"

# Initial memory allocation (MB)
memory = 128

# Number of VCPUs
vcpus = 2

# Disk Devices
# A list of `diskspec' entries as described in
# docs/misc/xl-disk-configuration.txt
disk = [ '/dev/vg/guest-volume,raw,xvda,rw' ]
```

The command used for managing the virtual machines is called xl. This command allows designers to manage the entire life cycle of Xen virtual machines from startup to shutdown. To create a new virtual machine instance based on an existing configuration file, the designer can create a Xen hypervisor with a simple one-line command:

```
xl create <path_to_configuration_file>
```

During the life cycle of the virtual machine, other `xl` options such as `list`, `reboot`, and `shutdown` can be used for management tasks.

Conclusion

Many types of systems can benefit from virtualization. Many system requirements that would otherwise be complex and highly labor-intensive can be implemented using Xen on multiprocessor platforms like the Xilinx Zynq UltraScale+ MPSoC featuring the ARMv8 processor. Using virtualization enables these desirable and oftentimes critical techniques:

- Partitioning
- Isolation
- Reliability
- Redundancy

Virtualization not only makes such systems possible; it can even make them easy. System designers ought to evaluate their designs in the light of using a virtualization solution to assure the best, most cost-effective results. The Xilinx Zynq UltraScale+ MPSoC, featuring the ARMv8 multiprocessor, is the ideal design platform for the Xen hypervisor and Xen toolset.

For software developers, virtualization enables development of software stacks that are portable and reliable, with minimal glue logic to enable them to perform as required. In addition, Xen is robust with the full weight of the Linux Foundation behind it and robust integration and support across a wide variety of operating systems and software stacks. Finally, managing Xen-based virtual machines is simple with an easy-to-use command line interface and plain ASCII text configuration files suitable for revision control.

The Linux kernel available from Xilinx on GitHub at https://github.com/Xilinx/linux-xlnx has support for Xen enabled. Source code for Xen hypervisor and the Xen tools, as well as more information about the Xen Project, can be found at http://www.xenproject.org/.

DornerWorks is Xilinx's partner for Xen hypervisor. See more information at: http://dornerworks.com/services/xilinxxen

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
03/31/2016	1.0	Initial Xilinx release.

Disclaimer

Automotive Applications Disclaimer