2016 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
VEHICLE ELECTRONICS AND ARCHITECTURE (VEA) TECHNICAL SESSION
AUGUST 2-4, 2016 – NOVI, MICHIGAN

XEN ON THE ZYNQ ULTRASCALE+ MPSOC

Robert VanVossen
DornerWorks, Ltd.
Grand Rapids, MI

ABSTRACT

One of the best ways to achieve full hardware utilization while maintaining a strict level of security and safety in a single System on a Chip (SoC) is through the use of virtualization. In this paper, we will explain the capabilities of the Xilinx Zynq UltraScale+ MultiProcessor SoC (MPSoC) and how they relate to target technology areas such as ARM processors and multi-core technology. We will also explain the features of Xen that aid in improving the safety and security of a virtualized system. We will provide examples of how to utilize these features, identify benefits, and explain how they can be used to implement several technology features including: SWAP-C reductions via consolidations, modular software architectures, and integration of multiple real-time operating systems.

INTRODUCTION

As embedded hardware improves, the added complexity makes it increasingly difficult to utilize all of a new platform's available computing resources while maintaining the same levels of safety and security. With the rise and expansion of nation-state funded cyber terrorism, safety and security are an ever growing concern for the U.S. Army and the USMC, and the demand for rigorous safety and security must be met for all systems. One of the best ways to achieve full hardware utilization while maintaining this strict level of security and safety in a system is through the use of virtualization.

A hypervisor is the foundational software that provides a means to virtualize a system. Xen is one well-established example that makes it possible for multiple commodity and real-time operating systems to be run concurrently in their own partitioned hardware space. Strict memory partitioning means that one guest operating system cannot read, write, or interfere in any way with the memory of another guest. Xen also has several features, such as CPU pinning and CPU-pools, that allow for fine grained control of the scheduling and processing time of these guests across the multiple cores. For example, CPU pinning allows for a guest to be run only on specified physical CPU cores, giving a guest guaranteed levels of performance and significantly reducing its ability to interfere with other guests elsewhere on the system. CPU-pools further enhance this control by allowing different scheduling algorithms to be applied to each of the system's configured pools of CPU cores.

ZYNQ ULTRASCALE+ MPSOC

The Zynq UltraScale+ MPSoC is one example of modern, powerful hardware that can use a hypervisor to manage its complexity. This system on a chip created by Xilinx contains several interconnected processing units, including a quad-core ARM Cortex-A53 application processor, a dual-core ARM Cortex-R5 real-time processor, and an ARM Mali-400 Graphics Processing Unit (GPU), all tightly coupled to the internal 16nm Xilinx UltraScale+ programmable fabric [1]. The ARM Cortex-A53 processor is known as the Application Processing Unit (APU) while the ARM Cortex-R5 is known as the Real-time Processing Unit (RPU). The ARM Cortex-A53 supports the 64-bit ARM specification, ARMv8, and the Xilinx Zynq UltraScale+ MPSoC is one of the first SoCs containing the Cortex-A53 processor to come to market.

The Z US+ MPSoC takes advantage of some of the newest ARM peripherals that improve virtualization, such as the Generic Interrupt Controller (GIC) and the System Memory Management Unit (SMMU). However, the complexity of this new system on a chip makes it very difficult to fully utilize with a single operating system or application, and further, running multiple applications concurrently presents security issues. This is where the Xen hypervisor comes in.

XEN

The Xen hypervisor is a mature, open source project that started as a research project almost 20 years ago and has been in production use for over 12 years. Xen started as the "XenoServer" in the late 1990's at the University of Cambridge. The name was changed to the "The Xen Project" in 2003 and Xen 1.0 was released the next year in 2004. In 2007, Citrix acquired XenServer and reaffirmed continued corporate sponsorship of The Xen Project and its open-source development. In 2013, a fully functional Xen port to ARM was released. Xen also joined the Linux Foundation that year.

Xen is a Type 1 hypervisor, which means that it runs directly on the hardware, as opposed to a Type 2 hypervisor that is layered above another Operating System (OS) [2]. A Type 1 hypervisor does not incur the overhead of the host OS, which makes it the most suitable option for embedded platforms. A Type 1 hypervisor gets fine grain control of all of the system resources, instead of just the resources that a Host OS would provide. A Type 1 hypervisor typically reduces the number of attack vectors compared to a Type 2 hypervisor.

Xen on the Zynq UltraScale+ MPSoC runs solely on the APU. This leaves the RPU open for non-virtualized applications. Since Xen adds some overhead (slight, but not zero), applications that have very strict timing requirements can be run independently on the RPU. A block diagram of an example Xen system running on the Z US+ MPSoC can be seen in Figure 1.

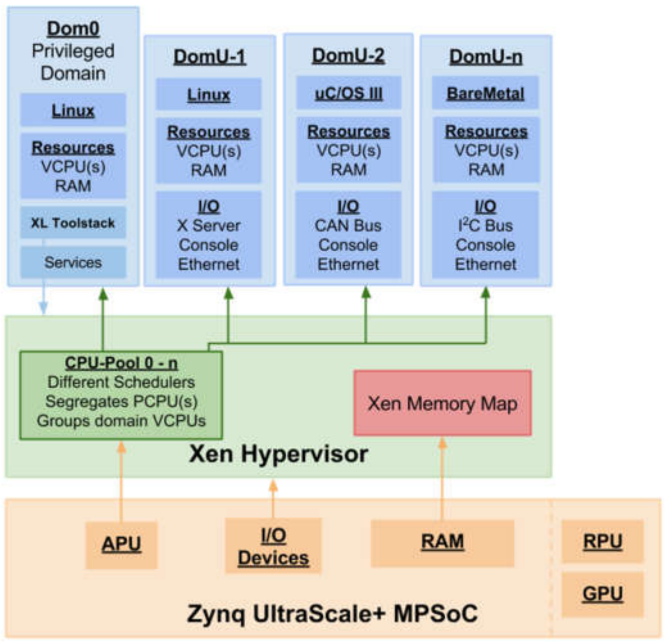

Figure 1: Xen on the Z US+ MPSoC

Benefits

The Xen hypervisor provides several benefits that should be considered when designing a system. (1) One of the main reasons to use a hypervisor is that it allows multiple OSes to be run on the same processor. This means that a legacy design that was spread among multiple, federated processors, can now be integrated on a single processor. This reduces the Size, Weight, and Power/Cost (SWaP-c) of the system. (2) Using the Xen hypervisor also improves code portability of applications/Operating Systems. Once an application or OS runs on top of Xen, it is much easier to port it to future hardware. Since Xen provides abstraction from the underlying hardware, old virtual machines can be easily migrated to Xen running on new hardware. (3) The isolation Xen provides between its guests also greatly enhances the security and safety of the system. For example, the Xen Inter-Domain Communication framework provides the potential for a design with red and black on the same system for a Cross Domain Solution (CDS). Xen has been used in SecureView, which is a CDS for x86 platforms. (4) Using Xen also increases the reliability of a system by providing redundancy. Redundant guests can run concurrently, so if one guest is comprised, the other guest can take its place.

ARM HARDWARE VIRTUALIZATION

The ARM processor architecture has a rich history back to the 1980's, and with the ARMv7a instruction set release in 2011, ARM processors have provided support for virtualization extensions. These extensions provide a means to make virtualization easier to perform with less required software. To understand how these extensions work, one must first understand exception levels. In the ARM architecture, an Exception Level (EL) is an operating mode that dictates which instructions can be executed and which registers can be accessed [3]. EL3 is the highest level, with full access to all processor functionality. This is where a secure monitor is expected to run. A secure monitor is the software in the system that has the potential to manage everything else. It has the highest privilege and therefore should be the most trusted piece of software. EL2 is where a hypervisor is expected to run. EL1 is where an OS is expected to run, with limited access to the processor. EL0 is where applications are expected to run, with limited access to the processor. Xen follows this suggested specification, running at EL2 and booting guest operating systems at EL1.

Each exception level has its own copy of certain registers. For example, each exception level has its own table for the Memory Management Unit (MMU). This allows Xen to map the memory space for a guest OS at Stage 2. Then the guest can map its own memory using Stage 1 translation tables, creating a virtual-to-physical mapping as it normally would, but the physical addresses are really just intermediate values that then go through the Stage 2 translation set up by Xen. This makes it easier to support the many possible memory configurations that could exist across multiple guest operating systems.

Some devices also have a virtualization-specific register set. One example of this is the system timer of the processor. A guest, which is running at EL1, will not be able to access the physical system timer, but it can access the virtual timer. The virtual timer works in almost the same exact way as the physical timer, but has an offset for each guest, so a guest cannot derive information about the hypervisor or other guests.

As the name suggests, Exception Levels correspond to the occurrence of exceptions. Each possible exception in the system occurs at a specific EL. For example, if the current exception level is EL1 and an address was not mapped in the EL1 MMU, then when that address is accessed, a data abort occurs and the exception handler at EL1 gets called and the system stays at EL1. If the current exception level is EL1, the address was mapped in the EL1 MMU, and that address was not mapped in the EL2 MMU, then a data abort occurs and the exception handler at EL2 gets called and the system transitions to the higher exception level, EL2. This means that any issues get handled at the correct level of software.

ARM also allows configuration for certain registers to be accessed at lower (less privileged) exception levels. Therefore, certain registers that are not deemed to be a security concern by Xen can be modified directly by the guest. One example of this is the "TLBI VAE1" (TLB Invalidate by VA, EL1) special register. Xen configures the system so that a guest can write to this register. This is not a security concern because the guest can only invalidate its own virtual memory mappings. In other cases, the register access can cause an exception at a higher Exception Level so that the software there can validate the request. One example of this is the "GICD_ICFGRn" (Interrupt Configuration Registers) register. Xen configures the system so a guest can not write to this register. Xen handles the interrupt that occurs and arbitrates the request. If a guest is allowed to map that interrupt, based on configuration, then Xen does the actual register write. If the guest is not allowed to map that interrupt, then Xen just returns to the guest. Xen uses these mechanisms to give the guests exactly the rights they need and no more (i.e., the security principle of least privilege).

INPUT/OUTPUT (I/O)

Embedded systems need to interact with various forms of input and output. Therefore, a method is needed to manage the I/O between the multiple guests. Xen provides two methods of I/O handling: paravirtualization and pass-through. The benefits and drawbacks of each method of device mapping will be described in this section.

Paravirtualization

I/O Paravirtualization uses software to share a device from a privileged guest to any other guest that needs to access the device. The privileged guest is the only one that has direct access to the device and contains the normal device driver to interact with the device. Then what Xen calls a split driver is used to share the data from the privileged guest to the other guests. A split driver is made up of a backend driver in the privileged guest and a frontend driver in the other guests that want to access the device. The backend driver sets up a shared ring buffer, and an event channel (a notification) for each guest that needs to access the device. The frontend driver in each guest then connects via a wrapper Application Program Interface (API) to those sharing mechanisms. A diagram of a split driver can be seen in Figure 2.

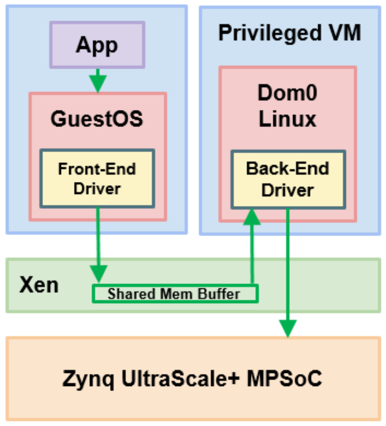

Figure 2: Paravirtualized split driver

Since the privileged guest arbitrates access to the device, the data from the device can be shared across virtual machines without breaking partitioning. This is useful if multiple guests need to access the same I/O channel. Another advantage is that the frontend driver presents an abstraction of the specific device, so that guests can be more generic and thus more portable. This can be an initial drawback, because if the guest OS does not support that frontend driver, it needs to be developed. Paravirtualization also adds another layer to the device driver stack, therefore the performance will not be as fast as native OS usage of the device. If multiple guests are sharing the same device the privileged guest must implement an allocation scheme to prevent a guest from monopolizing that device. Since this method takes advantage of the strict memory sharing infrastructure of Xen, it is a safe and secure method for handling I/O.

Pass-Through

Device pass-through uses the System Memory Management Unit (SMMU) and the GIC that is available on the Zynq UltraScale+ MPSoC to directly pass a device to a guest (Intel architectures provide a similar device called an IOMMU). Pass-through means that the guest gets sole access to the device. A diagram of a passed-through device can be seen in Figure 3.

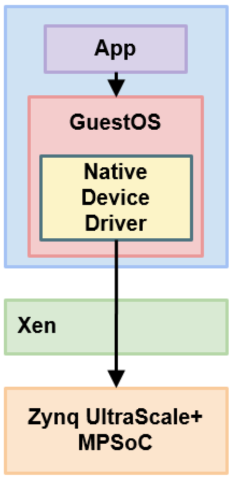

Figure 3: Passed-Through Device

This direct and exclusive access to the device is enforced by the SMMU hardware, making this I/O method quite secure. Only memory-mapped devices, such as a UART, can be passed-through. Also, bus-style interfaces must pass-through the root bus controller. For example, an entire SPI bus needs to be passed into a guest – not individual SPI devices. Pass-through provides the best performance, since the guest gets direct access to the device, but at the cost of preventing sharing of devices. The Z US+ MPSoC helps mitigate this by providing a large number of peripherals, such as 4 Ethernets and 4 UARTs. An example simple configuration of XZD is shown in Figure 4.

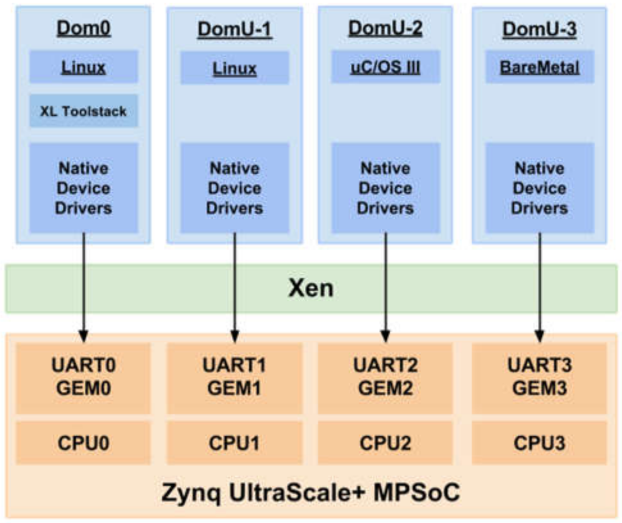

Figure 4: Simple Example XZD Configuration

This simple configuration with Xen on the Zynq UltraScale+ MPSoC could have system domain (Dom0) mapped to CPU0 with a UART and an Ethernet device, then 3 guests could each run on their own CPU with their own UART and Ethernet device passed-through. Xen is used to enforce the passed-through devices and CPU configurations. Pass-through can also be used to give a guest direct access to additional peripherals that are instantiated on the FPGA of the Z US+ MPSoC.

MULTICORE AND SCHEDULING

Since the Z US+ MPSoC contains a quad-core processor, it is important to effectively manage which tasks run on which cores. This requires schedulers that are multicore capable. Xen provides multiple schedulers that have this ability. Xen also provides a couple of ways to more finely control the system than just choosing the scheduler. These features are CPU Pinning and CPU Pools.

Schedulers

Xen has several available schedulers: Credit, ARINC653, and RTDS. These schedulers determine which guest runs on which physical Central Processing Unit (pCPU) using virtual CPUs (vCPUs), where the vCPUs are schedulable processor units. Each guest can have multiple vCPUs, where each vCPU is run on a pCPU. This means that a guest can run on multiple cores at the same time, allowing multi-threaded applications in one guest to take advantage of multiple cores.

The Credit scheduler is the default scheduler in Xen. It is a “fair time” algorithm similar to the default Linux scheduler. This scheduler gives guests a roughly equal amount of time, with some load balancing built in, therefore it is the scheduler that is used in the vast majority of server applications. This scheduler supports multicore scheduling. This scheduler maximizes CPU throughput at the cost of latency.

The ARINC653 is a real-time scheduler that meets the aviation standard, ARINC653, originally developed by DornerWorks and contributed to the Xen open source community. It is a scheduler that provides strict time isolation and system determinism. If a guest needs to run for 20 ms every 30 ms, that time is guaranteed to be given to that guest. This scheduler supports a limited form of multicore support using CPU Pools. The determinism of this scheduler does reduce the CPU throughput since it is not a work conserving algorithm.

The Real Time Deferrable Server (RTDS) scheduler, originally called RT-Xen [4], is an experimental real-time scheduler for the Xen hypervisor. vCPUs are scheduled using the deferrable server algorithm. A vCPU receives budget of CPU resources every period. Budget is replenished at the start of every period. Each vCPU consumes budget when running and suspends execution when no budget remains. The vCPU's budget is preserved when there is no task, i.e., another vCPU will not be scheduled until the budget is depleted for the current vCPU.

CPU Pinning

CPU Pinning is a configuration option that specifies the physical CPUs (pCPUs) on which a guest can run. This can be as simple as allowing a guest to run on a single pCPU or pinning the guest to a sequence of the available pCPUs. Pinning a guest to a pCPU does not limit any other guests from running on the pCPU. If a system needs a single critical guest and a few other less critical guests, CPU Pinning can be used to pin the critical guest to one or two pCPUs and then the other guests can be pinned to the remaining available pCPUs.

CPU Pools

CPU Pools are used to separate the physical CPUs into scheduler pools. This can be used to have more than one type of scheduler running on the system at the same time. This is useful if a system needs a real-time scheduler for just a portion of the guests. Each pCPU can only be in a single pool, but each pool can contain multiple pCPUs. Each pool can only run a single scheduler. CPU pools are used to achieve ARINC653 on a multicore system. Each CPU is placed in its own pool. Therefore, each CPU has its own instance of the scheduler running.

XEN ZYNQ DISTRIBUTION

The Xen Zynq Distribution (XZD) from DornerWorks provides a prebuilt Xen system that works on the Zynq UltraScale+ MPSoC. This distribution is highly customizable and completely open source. The distribution provides all of the necessary components for booting and using a Xen system: a Xen kernel, a sample Device Tree Blob (DTB), a system domain (dom0, including a Linux Kernel and File System), sample guest configuration files, a sample Linux guest kernel, sample Linux guest file systems, and a sample "Baremetal" guest image. The XZD is a means to quick start development or to easily test out Xen and its features on the Z US+ MPSoC. XZD also has a companion User Manual that explains how to boot the system, perform basic Xen administrative tasks, and how to build the XZD from source code. XZD can be downloaded for free from http://xen.world.

PERFORMANCE

One of the biggest concerns with embedded systems is performance. Adding anything new to a system can increase overhead to the system, which may prevent the system from meeting requirements.

Developers at Citrix executed performance benchmarks on the Applied Micro X-Gene, an ARMv8 64-bit 8 cores 2.4 Ghz processor, and on an Intel Xeon CPU X5650 [5]. In most cases, Xen on ARM had less overhead than Xen on x86. In all cases, Xen had less overhead than KVM (an open source competitor hypervisor to Xen). In all cases, Xen on ARM had a 2% or less virtualization overhead increase. That is, virtualizing applications incurred less than 2% performance penalty compared to running them natively.

Xen Overhead on ARM
< 2%

Table 1: Xen Overhead Percentage

Boot Time

DornerWorks has performed measurements of the boot time and interrupt overhead of Xen on the Xilinx ZCU102. Normally, to get performance numbers in a Xen system Xentrace is used. Xentrace is a series of Xen system calls that captures the current time and saves it for later viewing. Xenalyze is an application that is then used to read the results from Xentrace calls. However, Xentrace is not available for use at the entry point of Xen or earlier, so a different method was needed to get the boot times. The code was instrumented at key points in the boot-up sequence and timestamps were collected at those points. Each time stamp was collected with a single assembly instruction. These timestamps are displayed on the console after the boot-up sequence has completed, so that it does not influence the boot time. The timer used to get the time stamps is the 64-bit ARM Physical Generic Timer. This timer has a frequency of 100 MHz (10 nanosecond resolution). The boot sequence of XZD starts with the First Stage Boot Loader (FSBL), which sets up peripherals like the

DDR controller and programs the FPGA with a Bitstream. The FSBL then executes U-Boot, which loads the Xen kernel, the system DTB, and the Dom0 Linux kernel. U-Boot then executes the Xen kernel. The Xen kernel initializes and then executes Dom0. For these measurements, we start at U-Boot because the boot time of the FSBL can vary due to different booting methods and configuration. Using this methodology, the boot times at the start of U-Boot were found as follows:

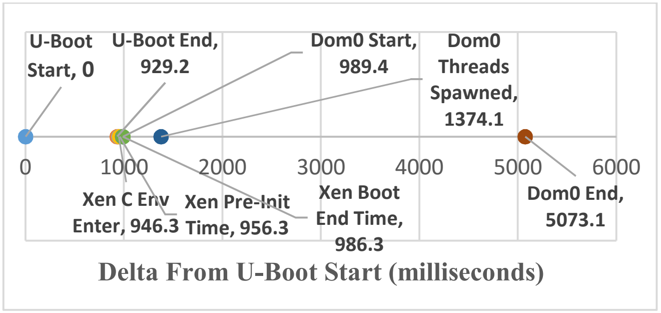

Figure 5: Xen System Boot Times

U-Boot ends booting after 929.2 ms and the Xen kernel finishes booting after 986.3 ms. So Xen itself only adds 57.1 ms to the boot time. Dom0 currently adds another 4086.8 ms to the boot time. The Dom0 kernel that was used for testing was a Linux Ubuntu distribution, which could likely be optimized to remove unused drivers. This would help reduce the boot time of the Dom0 kernel.

Interrupt Overhead

The approach taken for determining the hypervisor impact to IRQ latency was to measure the average IRQ latency of an application running natively, *delay*, and then subtract that from the average IRQ latency of the same application running as a Xen guest, *delay'*. This approach factors out any hardware or application software delays and leaves only the additional delay that should be attributed to running the application as a Xen guest. To measure delay and delay' an application was written that, upon detection of a general purpose I/O (GPIO) line going low, changes the state of a second GPIO line. The ZCU102 provides a PS push button on MIO-22 and a PS controlled LED on MIO-23, which were used to meet the application's GPIO needs. The application toggles the LED state almost immediately after the interrupt generated by the push button vectors to the generic IRQ handling code. Only a minimal context, specifically registers X0 and X1, is saved off before toggling the LED. The PS push button was pressed and an oscilloscope was used to measure the time delta between the start of the waveform change on the PS push button channel to the start of the waveform change on the LED channel. Using this process, the following data sets were collected.

Sample	Native Linux IRQ Latency (usec)	Xen Linux Guest IRQ Latency (usec)
1	0.272	2.56
2	0.272	2.7
3	0.636	2.54
4	0.644	2.6
5	0.272	2.34
6	0.272	2.66
7	0.272	2.52
8	0.272	2.58
9	0.272	2.64
10	0.272	2.8
11	0.272	2.54
12	0.272	2.56
13	0.284	2.68
14	0.264	2.58
15	0.264	2.68
16	0.272	2.54
17	0.268	2.66
18	0.268	2.62
19	0.268	2.52
20	0.252	2.56
21		2.52
22		2.56
23		2.56
24		2.44
Average (usec)	**0.307**	**2.58**
Xen IRQ Delay (usec)		**2.27**

Table 2: Interrupt Overhead Samples

The Xen delay was calculated as follows:

Xen delay = delay' - delay

Xen delay = 2.58usec - 0.31usec

Xen delay = ~2.3usec

CONCLUSION

As computing systems become more complicated, hypervisors can be used to more easily utilize all the power a platform provides. The Xen Project is a mature, open source hypervisor which has been used in the industry as a secure, safe, and efficient solution. Using the hardware virtualization extensions that the ARM Cortex-A53 processor provides, Xen provides a virtualized environment to run multiple guests on the single chip. The Z US+ MPSoC provides an SMMU and a GIC, which allows Xen to pass-through I/O devices to guests to give them sole, direct access to those devices. All of these features combined with Xen implementation can be used to bring virtualization to the embedded world.

The Xen Zynq Distribution provides a means to quickly get up and running with Xen on the Zynq UltraScale+ MPSoC. This distribution can be modified to meet the needs of the user and is a great way to start a project (http://xen.world). It is also useful to test out the Xen hypervisor.

REFERENCES

[1] Xilinx, “Zynq UltraScale+ MPSoC Technical Reference Manual - UG1085 (v1.1)”, March 2016.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the Art of Virtualization”, ACM SIGOPS Operating Systems Review, 2003.

[3] ARM, “ARMv8 Architecture Reference Manual - A.g”, July 2015.

[4] S. Xi, M. Xu, C. Lu, L.T.X. Phan, C. D. Gill, O. Sokolsky and I. Lee, “Real-Time Multi-Core Virtual Machine Scheduling in Xen”, ACM International Conference on Embedded Software (EMSOFT'14), October 2014.

[5] S. Stabellini, “Xen on ARM: status and performance”, Xen Developers Summit, August 2014.