energies

MDPI

Article

Evaluating Latency in Multiprocessing Embedded Systems for the Smart Grid

Sara Alonso *,†, Jesús Lázaro, Jaime Jiménez, Unai Bidarte and Leire Muguira

Universidad del País Vasco/Euskal Herriko Unibertsitatea (UPV/EHU), 48013 Bilbao, Spain; jesus.lazaro@ehu.eus (J.L.); jaime.jimenez@ehu.eus (J.J.); unai.bidarte@ehu.eus (U.B.); leire.muguira@ehu.eus (L.M)

* Correspondence: sara.alonso@ehu.eus

† Current address: Plaza Ingeniero Torres Quevedo, 1, 48013 Bilbao, Spain.

Abstract: Smart grid endpoints need to use two environments within a processing system (PS), one with a Linux-type operating system (OS) using the Arm Cortex-A53 cores for management tasks, and the other with a standalone execution or a real-time OS using the Arm Cortex-R5 cores. The Xen hypervisor and the OpenAMP framework allow this, but they may introduce a delay in the system, and some messages in the smart grid need a latency lower than 3 ms. In this paper, the Linux thread latencies are characterized by the Cyclictest tool. It is shown that when Xen hypervisor is used, this scenario is not suitable for the smart grid as it does not meet the 3 ms timing constraint. Then, standalone execution as the real-time part is evaluated, measuring the delay to handle an interrupt created in programmable logic (PL). The standalone application was run in A53 and R5 cores, with Xen hypervisor and OpenAMP framework. These scenarios all met the 3 ms constraint. The main contribution of the present work is the detailed characterization of each real-time execution, in order to facilitate selecting the most suitable one for each application.

Keywords: virtualization; Xen hypervisor; system-on-chip; latency; Cyclictest; interrupt; OpenAMP; multiprocessing

Citation: Alonso, S.; Lázaro, J.; Jiménez, J.; Bidarte, U.; Muguira, L. Evaluating Latency in Multi-Processing Embedded Systems for the Smart Grid. *Energies* **2021**, *14*, 3322. https://doi.org/10.3390/en14113322

Academic Editor: Joao Ferreira

Received: 30 March 2021
Accepted: 2 June 2021
Published: 5 June 2021

1. Introduction

Multicore systems are currently very popular, due to the possibilities they offer [1]. On the one hand, in a homogeneous multicore, all the CPUs have the same architecture. They are used when the application needs more CPU power to manage its workload. On the other hand, in a heterogeneous multicore, the cores may have different architectures and they allow having different OSs in different cores.

Smart grid endpoints need to have a hybrid node where real-time tasks and non-real-time ones coexist in the same device. There are two solutions for this: virtualization, which is used in homogeneous multicores, and multicore frameworks, which are used in heterogeneous multicores.

Virtualization requires a hypervisor, a layer of software used as an interface between the device and the OS. It controls virtual machines and allots resources to the physical platform on which it is installed [2]. Hypervisors are commonly used in multicore embedded platforms [3–12]. Xen is a widely used one on system-on-chip (SoC) platforms.

Virtualization has multiple benefits. First, as it enables sharing of physical resources, the cost is highly reduced in electronics and, therefore, for power consumption. Second, it increases the system's security, thanks to its ability to isolate the virtual machines (VMs). Moreover, it eases the management of the tasks and allows the user to exploit the desired computational resources [13].

However, hypervisors do not allow for using the general-purpose and real-time processing cores at the same time. Therefore, if hard timing constraints are needed, a multicore framework is another solution. It supports having an OS in the general-purpose

cores, and another OS in real-time cores, facilitating the management between them. In this way, real-time tasks can be run on a more deterministic core, while a Linux OS can manage them from the other cores. However, the integration of Xen or OpenAMP in the architecture could also introduce a delay [14]. OpenAMP is a widely used multicore framework on SoC platforms.

In a smart grid, it is mandatory to achieve latency of less than 3 ms [15] when generating, transmitting and processing generic object-oriented substation events (GOOSEs) and sampled value (SV) messages in electrical substations. Therefore, it is essential to characterize the latencies introduced by the Xen hypervisor and the OpenAMP framework and check which one meets this time requirement better.

In this context, Cyclictest is executed, which is a tool to measure the difference between a thread's intended wake-up time and the time at which it actually wakes up, in four scenarios: (1) a Linux OS running on the four A53 cores, (2) a Linux OS running on the four A53 cores with stress, (3) a Linux OS running on Xen's DomU and (4) a Linux OS running on Xen's DomU with stress in Dom0. It is shown that these scenarios are not suitable for real time, so another test scenario has been evaluated, where the latency of handling, in the PS part, an interrupt generated on the PL side is measured. The GPIO interrupt latencies in six scenarios are: (1) a standalone application running on an A53 core, (2) a standalone application running on an R5 core, (3) a standalone application running on a Xen domain while a Linux OS is running on Xen Dom0 without stress, (4) a standalone application running on an R5 core while a Linux OS is running on the A53 cores using OpenAMP without stress, (5) a standalone application running on a Xen domain while a Linux OS is running on Xen Dom0, stressing it, and (6) a standalone application running on an R5 core while a Linux OS is running on the A53 cores using OpenAMP stressing the A53 cores. Then, we evaluated if these scenarios meet the 3 ms latency constraint.

Some related works are found in [16–18]. The first and second studies measure and compare the latency introduced by virtualization with the Cyclictest tool. In the first part of this article, similar measurements are made with similar results. The other study measures some latencies introduced by OpenAMP, as is carried out in the second part of this study. However, they use different tools and measure different latencies to the ones analyzed in this work. Our work adds the connection between PL and PS and evaluates latency for complex systems with real-time FPGA circuits connected to a multiprocessor system managed by a hypervisor or multicore framework. The multiprocessor is composed of real-time software and management software on Linux. This way, we make precise measurements based on an embedded logic analyzer. The study concludes that it is feasible to use these systems for a smart grid, even for the most critical communications.

Below, Section 2 describes the multicore embedded systems, Section 3 describes the Xen hypervisor and the OpenAMP framework, Section 4 shows the methodology and tools used for the measurements, Section 5 summarizes the results and Section 6 concludes the article.

2. Multicore Embedded Systems

Symmetric multiprocessing (SMP) systems have multiple CPUs with the same architecture. They are always homogeneous multicore designs. The CPUs share all or a part of the memory space. Typically, a single OS is used, running on all the CPUs, so that the work is divided between them. Therefore, SMP is usually used when an embedded application needs more CPU power to manage its workload.

An asymmetric multiprocessing (AMP) system has multiple CPUs, each of which may have a different architecture. Each has its own address space, and each may run a different OS. AMP is typically used when different CPU architectures are needed for specific tasks. In an AMP system, a different OS can be run on each core as the task requires.

Multiprocessor system-on-chip (MPSoC) devices combine, on the one hand, a PS part with Quad Arm Cortex-A53, for general-purpose application processing, and Dual Arm

Cortex-R5F, for real-time applications processing, and, on the other hand, a PL part with an FPGA. This makes these devices flexible and powerful.

In a smart grid, the generation, transport and use of some data must be carried out with deterministic characteristics. When the equipment performs data switching tasks using MPSoC-type circuits, the data routing at high speed and in real time is carried out by employing specific circuits in the PL part. Usually, the management and configuration of these IPs are carried out through software executed within general-purpose OSs in the PS.

Endpoints are circuits that receive and transmit information and, in addition, have to make use of it through deterministic software. When they are developed, the previous configuration is no longer valid since this software cannot be executed within Linux-type OSs. In these cases, the software architecture becomes more complex. It is necessary to use two environments within the PS, one with a Linux-type OS using the A53 cores for management tasks, and the other one with a standalone execution or with a real-time OS, such as FreeRTOS, using the R5 cores. This is the scenario contemplated in the present work. The need for both software environments on a single MPSoC requires higher-level software to perform monitoring tasks.

When a hybrid embedded system is needed, there are two main possibilities: virtualization, if the architecture is SMP, and multicore frameworks, if the architecture is AMP. In this paper, the Xen hypervisor is used for evaluating the latencies introduced when using the virtualization technique, and OpenAMP is used as a multicore framework.

Therefore, hypervisors and multicore frameworks are helpful to have two different OSs in the same device. This facilitates the coexistence of real-time and non-real-time tasks. However, this software may increase the latencies, and a smart grid needs very specific time restrictions for some applications.

The International Electrotechnical Commission (IEC) has developed the IEC 61850 standard to overcome the interoperability issues in automated substations in industrial and measurement equipment from different vendors [15]. Layer-2 messages are used to provide services that require delivering high-speed (low-delay) messages [19]. Depending on the type of messages and their application, the maximum time requirement is set from 3 ms to more than 500 ms. GOOSE and SV messages are the most critical messages and must be generated, transmitted and processed in less than 3 ms even in worst-case scenarios.

The latency measures presented in this work attempt to assess compliance with this time requirement in equipment where three data processing environments exist in parallel:

- Electronic circuits in PL where the latency is usually fixed or limited.
- Software executed on real-time processors (R5).
- Software executed on processors not prepared for real time (A53) and within OSs without deterministic characteristics (Linux).

The use of hypervisors and multicore frameworks may have drawbacks, such as a significant code footprint, some execution overhead or the need for hardware virtualization in the processor in the case of hypervisors [20]. One of the most relevant disadvantages is the introduction of latencies. They are defined as the difference between the theoretical schedule and the actual one. Latencies introduced by the OS have been widely studied [21]. However, latencies introduced by hypervisors [22–25] or multicore frameworks [18] have often been overlooked.

3. Xen Hypervisor and OpenAMP Framework to Secure Smart Grid Multiprocessing Endpoints

3.1. Multicore Frameworks and OpenAMP

In heterogeneous multicore architectures, it is helpful to have a framework to handle the interaction of OSs in life cycle management, inter-operating system messaging, power management and resource sharing [26].

Multicore frameworks support an AMP multicore architecture and have a very small memory footprint and execution time overhead. Nevertheless, core workloads are not isolated from each other and it is difficult to control and debug the boot sequence [20]. The

main objectives of these frameworks are inter-core communication using remote processor messaging (RPMsg) and remote processor life cycle management using remoteproc to facilitate control of the boot order. OpenAMP is an extended multicore framework in embedded systems.

OpenAMP is a framework developed by the Multicore Association (MCA). It provides the software components needed to develop software applications for AMP systems, such as life cycle management (LCM) and inter-processor communication capabilities for managing remote computer resources. It also provides a library that can be used with a real-time operating system (RTOS) or standalone software environments.

OpenAMP uses the Libmetal library to access devices, handle interrupts and request memory across different operating environments. It is available for Linux, FreeRTOS and bare-metal systems. The OpenAMP library provides the implementation for RPMsg, virtualization module (VirtIO) and remoteproc, which are implemented in an upstream Linux kernel [27]. The VirtIO module manages the shared memory. It is a virtualized communication standard for network and disk device drivers. It provides a communication interface for upper-layer software by virtualizing the slave device [18]. Remoteproc is a software interface module that allows the LCM of the slave processors. It allocates system resources and creates virtIO devices. RPMsg is a software interface that provides inter-process communications (ICP) between kernel drivers and remote processors.

One leading technology for AMP architectures is dual-core shared resources. Another one is the dual-core boot method, which means that the slave processor is booted under the control of the master processor. The last one is the dual-core communication method, which means that dual cores have the right to read and write to the same memory area [28].

3.2. Virtualization and Xen Hypervisor

Virtualization consists of a representation of some hardware resources so that different guests can share them. Thus, resources and energy consumption are saved and the management of different applications becomes easier. In addition, it enables easier and faster application migration and the secure placement of antagonistic applications [29].

There are three virtualization levels: complete or hardware virtualization machine (HVM), para-virtualization (PV) and static partitioning or core virtualization [30], which consists of a combination of the previous ones. On the first level, it is possible to run an OS inside a virtual machine, as the hardware architectures have the needed support for virtualization. However, when HVM is not supported by the platform, PV is used. To run a PV guest, some of the privileged routines and instructions of the OS kernel need to be replaced by calls to the hypervisor. Although most hardware allows virtualization, PV is still used for some applications to get a better performance [16]. Owing to the restrictions of computing resources in embedded systems, static partitioning is the most suitable virtualization for them [14].

Virtualization requires a virtual machine monitor (VMM) or hypervisor. This is a software layer used as the interface between the OS and the physical device. It handles the distribution of physical resources and manages the guest machines or guest OS. Some other reasons for using a hypervisor in embedded systems are increasing security levels by isolating the different VMs and having different OSs on a single device.

There are two kinds of hypervisors: type 1 and 2. On the one hand, type 1 hypervisors run directly on hardware. They host an OS and handle resources and memory allocation for the VMs. On the other hand, type 2 hypervisors run over a host OS, which provides drivers and services for the hypervisor hosting a virtualized guest OS. Type 1 hypervisors give higher performance and versatility and type 2 hypervisors offer simpler management but lower efficiency.

However, a hypervisor supposes the integration of a new layer in the architecture, which may affect the system's capabilities when responding to events, and it may involve higher latencies [14]. Nonetheless, measuring the performance when building the virtual-

ization is not a simple task. Different OSs need to be installed, the hardware and software need to be configured and the merit figures need to be measured.

There are multiple hypervisors in the market, such as Xen, KVM or Jailhouse. Xen is widely used in embedded systems [16] and is open source. Furthermore, it enhances the security and reliability of the system. It is a type 1 hypervisor and uses a combination of hardware virtualization and para-virtualization to achieve the best performance. For example, while HVM technology is used for CPU and memory, I/O is often handled using PV [16].

Each VM has one or more virtual CPUs (vCPUs) assigned and each vCPU is seen as a single physical CPU core by the VM's OS. Xen's Credit Scheduler synchronizes all the vCPUs. This is a fair share algorithm based on proportional scheduling [31]. A specific credit value is assigned to each domain depending on its weight. Thus, if each domain is given the same number of credits, they will have the same fraction of processor resources. The credits of the running domains are deducted periodically. When the credit value is negative, the domain is over priority. Otherwise, it is under priority. Xen usually allocates just one vCPU to each domain when this is carried out and it contains the information related to scheduling and event channels.

Xen guests or virtual machines are called domains. Each guest runs its isolated OS and applications. Dom0 is the first domain created by the Xen hypervisor and it runs a Linux OS as a special PV guest. Xen uses the device tree to see the hardware and loads the corresponding drivers. When it finds the hypervisor node, Dom0 can initialize the needed backends. This domain is privileged, since it can access physical resources and manage other domains with XenControlTools. The rest of the domains are called DomU and do not have privileges. They can run any OS that has been ported to Xen, but they can only be created while Dom0 is running [16]. These domains cannot directly access the physical hardware on the machine. In an I/O operation, DomUs cooperate with Dom0 by using a ring buffer for the packet transmission and another one for the packet reception. Xen implements these based on event channels and grant tables. The latter ones are mechanisms to share or transfer memory pages between domains. Xen saves the grants in a reference table and passes the grant references to other domains. It signals these I/O requests by the event channels using hypercall. The target domain checks the event channels when it is scheduled and delivers the pending event by invoking the corresponding interrupt handler [31].

Some studies concluded that Xen introduces much larger latencies than other hypervisors, and these latencies are not due to the scheduler. The authors of [16] state, on the one hand, that the high latency inside Xen guests is related to the mechanism that routes the timer interrupts from the hypervisor to the guest. On the other hand, it says that Xen enforces a minimum period for the timer interrupts. Therefore, the delay is generated by overhead in the interrupt forwarding mechanism and the mechanism used to handle the guest's timers. In this paper, the latency of handling an interrupt is characterized in different scenarios with a standalone application in OpenAMP and Xen, after showing that Linux thread latencies do not meet the 3 ms constraint with Linux in Xen DomU.

4. Proposed Methodology to Measure Latencies Introduced by Xen When Time-Critical Software Runs in Linux

To measure the latencies, the scenarios in Figures 1 and 2 have been implemented on the ZCU102 board, based on a Zynq UltraScale+ MPSoC. This board is populated with the Zynq UltraScale+ XCZU9EG-2FFVB1156E MPSoC which combines a powerful PS and PL into the same device. The PS block has three major processing units: Cortex-A53 application processing unit (APU), Cortex-R5 real-time processing unit (RPU) and Mali-400 graphics processing unit (GPU). The Zynq UltraScale+ MPSoC also has a DDR memory controller and high-speed serial I/O (HSSIO) interfaces, supporting some protocols, such as USB 3.0. In addition, the PS-side Gigabit Ethernet MAC (GEM) implements a 10/100/1000 Mb/s Ethernet interface.

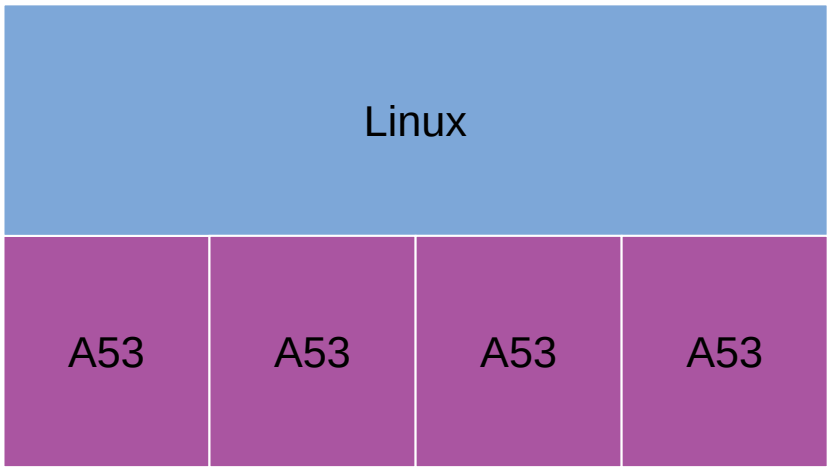

Figure 1. Scenario 1: Linux OS.

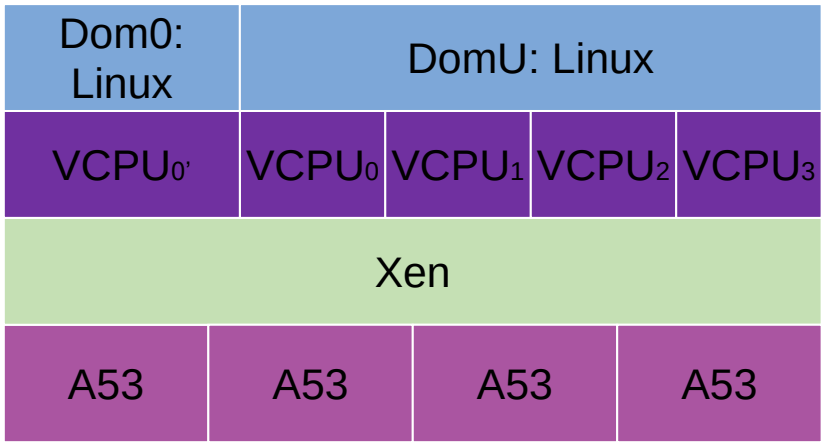

Figure 2. Scenario 2: Xen Linux Dom0 + Linux DomU.

The scenarios have been built with Petalinux tools [32]. Petalinux tools allow for installing the Xen hypervisor on the board and configuring it. First, a Petalinux project was created using the corresponding board support package (BSP) for the board. Then the Xen hypervisor was enabled in the root file system (rootfs) configuration and included in the device tree. Dom0's arguments indicate that this domain is run in a virtual console called hypervisor virtual console 0 (hvc0). Dom0 is allocated in the $0 \times 80{,}000$ memory address. The DomU is created from Dom0. The stress-ng, a tool to stress the system, and the rt-tests, a group of programs to test different real-time Linux features, must also be included in order to be able to use them later.

Cyclictest is part of the rt-tests packages, and it measures the scheduling latency accurately and repeatedly [33]. It can measure latencies in real-time systems caused by the hardware, the firmware and the OS. It is a Linux tool, so it can just be used in a Linux OS. It starts executing a non-real-time master thread, and this starts some threads with a real-time priority. The real-time priority threads are woken up by an expiring timer periodically. The calculated latency is the difference between the programmed and effective wake-up times.

It is interesting to characterize the latencies in a stressful situation to see the worst-case latency. Stress-ng stresses the system in various ways. It exercises some physical subsystems and OS kernel interfaces. It also has some CPU-specific stress tests.

In scenarios 1 and 2, Cyclictest was run. In the first scenario (Figure 1), Linux OS is running on the four A53 cores. In the second scenario (Figure 2), the Xen hypervisor is installed on the four A53 cores and then distributes the VCPUs as the user demands. Each core can create 8 VCPUs, so 32 VCPUs can be created in total in this scenario. In this case, in the second scenario, one VCPU is given to Dom0, and DomU is executed on Xen, and it

uses four VCPUs. Dom0 distributes the physical resources to the different VMs or domains, so it needs to be created before the others. Each DomU is a VM where user applications can be run.

Cyclictest has been run for the scenarios above, with one measurement thread per core, each running as SCHED_FIFO priority 95, and with all memory allocations locked. The SCHED_FIFO scheduling class is a real-time feature specified by a portable operating system interface (POSIX). The processes of this class can use the CPU for as long as they want, but they depend on the needs of higher-priority real-time processes. Each measurement was made for 24 h without stress, and while stress-ng was stressing the system with four overloads exercising the CPU sequentially through all the methods. While Cyclictest runs on Xen DomU, the stress-ng command runs in Xen Dom0 to stress the kernel.

5. Results When Time-Critical Software Runs in Linux

Tables 1 and 2 summarize the results of the maximum and average Linux thread latencies measured with Cyclictest for scenarios 1 and 2, with and without stress. The Linux thread latency is significantly bigger in the scenarios with the hypervisor. In addition, the stress increases that latency. It is also remarkable that hypervisors increase the average delay and the maximum, reaching very high peaks. Figures 3 and 4 show the results of Tables 1 and 2 graphically.

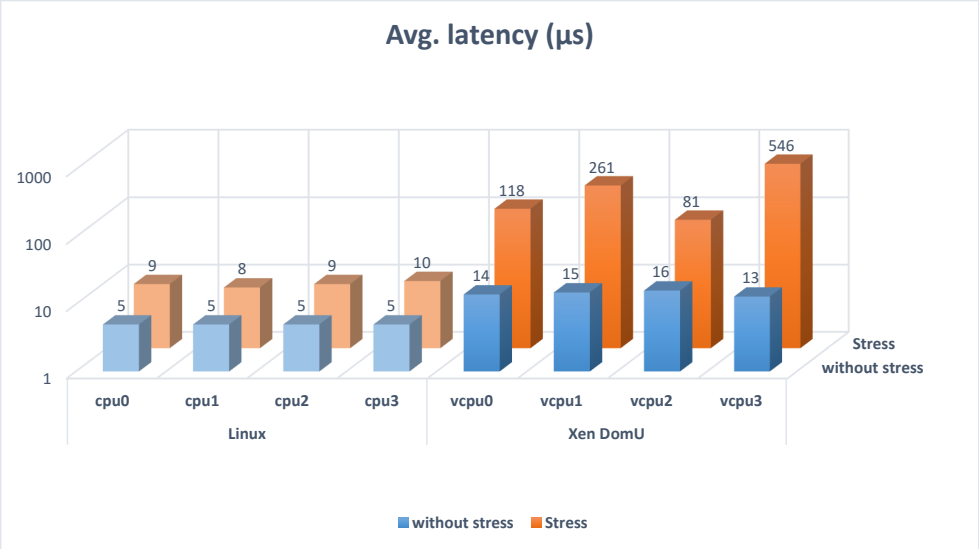

Figure 3. Average thread latency in each CPU and vCPU in Linux and Xen DomU with stress and without stress.

Table 1. Linux thread latency in Linux (Figure 1).

	Without Stress		With Stress	
CPU	**Avg. (µs)**	**Max. (µs)**	**Avg. (µs)**	**Max. (µs)**
0	5	69	9	177
1	5	63	8	250
2	5	44	9	129
3	5	11	10	67
Average	5	47	9	156

Table 2. Linux thread latency on Xen DomU: Linux (Figure 2).

	Without Stress		With Stress	
vCPU	**Avg. (µs)**	**Max. (µs)**	**Avg. (µs)**	**Max. (µs)**
0	14	275	118	109,592
1	15	129	261	103,871
2	16	104	81	102,803
3	13	128	546	103,520
Average	15	159	252	104,947

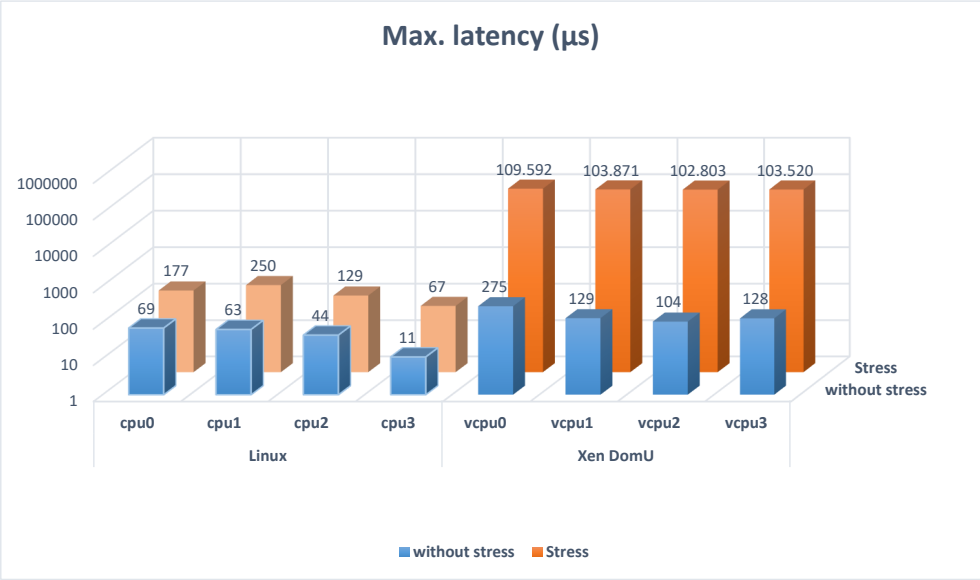

Figure 4. Maximum thread latency in each CPU and vCPU in Linux and Xen DomU with stress and without stress.

On the one hand, these graphics show that the average latency is double in Linux with stress than in Linux without stress and it is multiplied by 10 when the hypervisor is used in a stress situation. On the other hand, they show that the maximum latency is triple in Linux with stress than in Linux without stress. It is multiplied by 1000 when the hypervisor is used and the system is stressed. This high increase in the Xen DomU with stress situation could be problematic for some applications, limiting the processes where the latency must be controlled. When this latency is too high, not all the tasks are schedulable. Nevertheless, the worst-case latency is critical for a real-time system, so the significant increase in the maximum latency when the hypervisor is used in a stress situation is meaningful information.

The worst-case latency of this scenario does not meet the 3 ms requirement for GOOSE and SV messages in the smart grid. The authors of [34] present similar measures and also conclude that Xen, with its default configuration, is not suitable for real-time applications. Therefore, the interrupt handling delay has been measured in scenarios that use standalone as the real-time part, instead of Linux.

6. Proposed Methodology to Measure Latencies Introduced by Xen and OpenAMP When Time-Critical Software Runs as Standalone

The scenarios in Figures 5–8 have been implemented on the same board as scenarios 1 and 2.

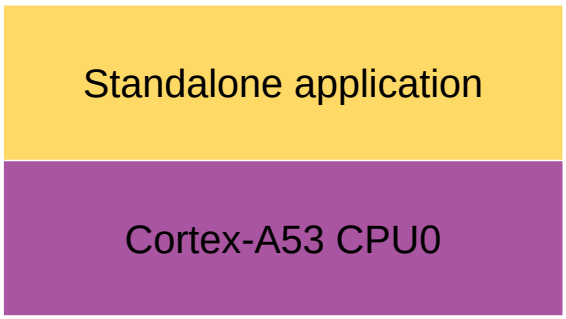

Figure 5. Scenario 3: standalone A53.

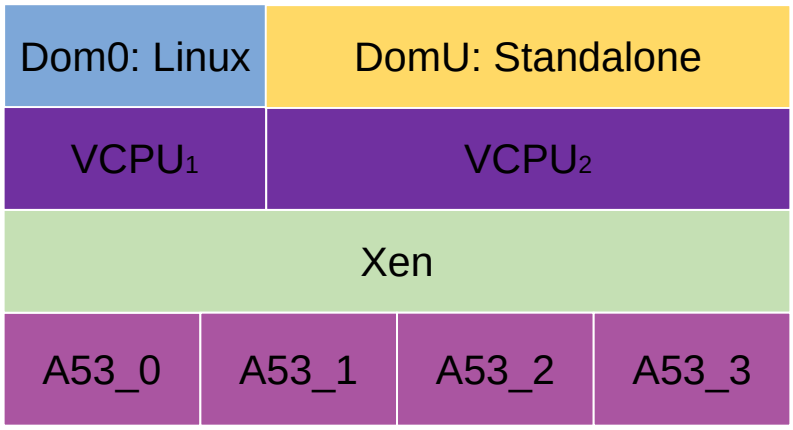

Figure 6. Scenario 6: Xen Linux Dom0 + standalone DomU.

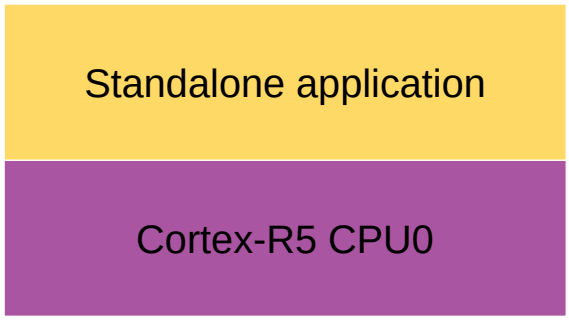

Figure 7. Scenario 4: standalone R5.

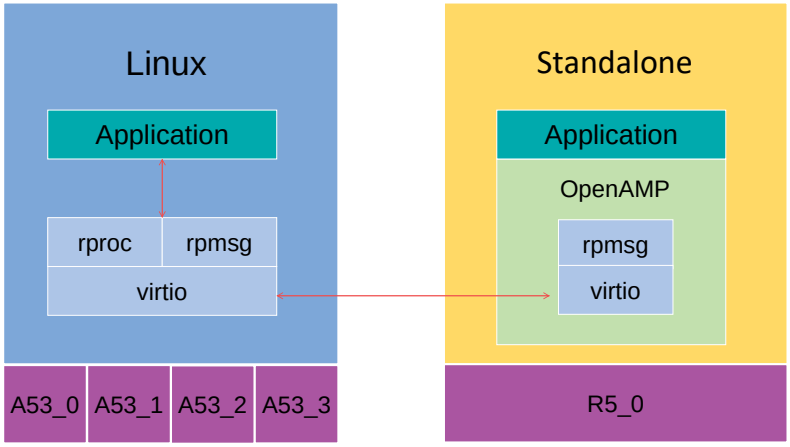

Figure 8. Scenario 5: OpenAMP Linux A53 + standalone R5.

In these scenarios, the GPIO interrupt latency has been measured from the PL side of the board. A hardware design has been made with a PS block, an AXI GPIO and an AXI Timer in Vivado (Figure 9):

1. In the PL part, the axi_timer_1 generates an interrupt.
2. This interrupt is included in the pl_ps_irq0 input of the PS block.
3. In the PS part, when the interrupt of the second step is handled, the axi_gpio_0 output signal is set to a logical one.
4. The output signal of the axi_gpio_0 block and the interrupt signal generated by the axi_timer_1 are connected to the embedded logic analyzer, so that when they are debugged, they can be displayed in the hardware manager.

Then, inside the microprocessor, in the application that is executed in the Cortex-R5 core, the following steps are carried out:

1. Initialize all the peripherals.
2. Handle the interrupt when it is set.
3. In the interrupt service routine (ISR), the first action is writing in the axi_gpio_0. Therefore, when the interrupt is handled, the axi_gpio_0 output signal is set to a logical one.

Then the application is executed in each scenario. We see in the hardware manager the output of the axi_gpio_0, written in the ISR, being triggered a specific time after the interrupt signal has been activated. Thus, we can calculate the latency between the activation of the interrupt signal generated in PL and the moment this signal is handled in the PS.

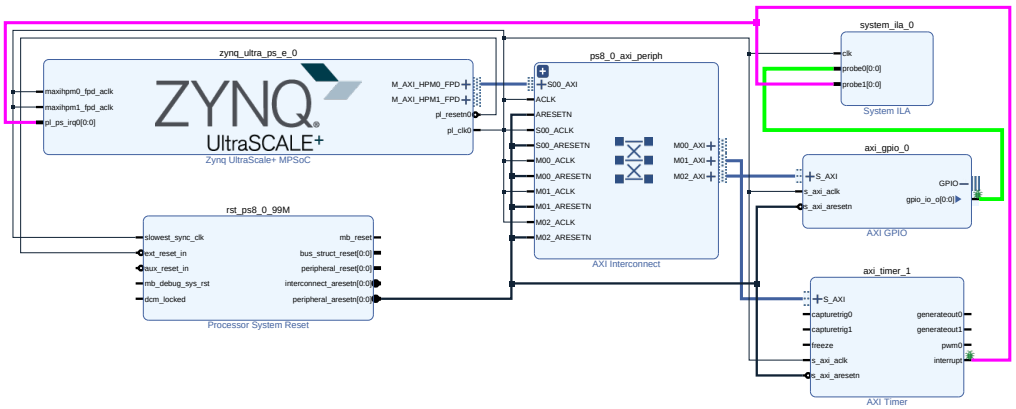

Figure 9. Hardware IP block design in Vivado.

These measurements were carried out in four different scenarios ,as shown in Figures 5–8. The two last scenarios have been evaluated in a non-stress and in a stress situation. For stressing the system, the stress-ng tool was used. The system was stressed from the management part (Dom0 in the case of Xen and the Linux OS in the case of OpenAMP). The system was stressed with four overloads exercising the CPU that worked sequentially through all the methods for overloading the CPU.

In the first scenario, there is not an OS, and the application runs directly on the hardware, on a Cortex-A53 core (Figure 5), the general-purpose one. In the second, the application runs directly on the hardware too, on a Cortex-R5 core (Figure 7), the real-time processing one. In the third, a Linux OS runs on the four Cortex-A53, which are the cores used for general-purpose application processing, while the bare-metal application runs on one of the Cortex-R5s (Figure 8). In the fourth scenario, Xen runs on the four Cortex-A53 cores, and a Linux OS runs on Xen Dom0, while the bare-metal application runs on another domain (Figure 6).

The scenarios with Xen and OpenAMP have been built with Petalinux tools. Xen is configured in the same way as in scenarios 1 and 2. For the scenarios with OpenAMP,

this framework must also be enabled and included in the device tree. When configuring OpenAMP, the application's executable must be added to the project, in order to run it later in the remote processor. The communication with the R5 core and its execution were managed from the Linux OS running on the A53 cores. The stress-ng tool must also be included in the Petalinux project.

When some determined time constraints are needed, interrupts are essential. Interrupts are commonly used in many utilities of the MPSoC boards and measuring them can be helpful in many applications. Additionally, it is interesting to characterize the latencies in a stressful situation to see the worst-case latency. Stress-ng forces the system in various ways. It exercises some physical subsystems and OS kernel interfaces. It also supports some CPU-specific stress tests. The stress-ng command runs in Xen Dom0 to stress the kernel in the scenario of Figure 6 and in the Linux part in the scenario of Figure 8.

7. Results When Time-Critical Software Runs as Standalone

Table 3 summarizes the average, standard deviation and maximum values of measured GPIO interrupt latencies. The measurement has been done 30 times in each scenario. Figure 10 shows the violin diagram of these results.

Table 3. Interrupt latency for 3 to 6 scenarios (Figures 5–8).

Scenario	Avg. (µs)	Dev. (µs)	Max. (µs)
Real-time software running on A53			
Standalone A53	1925	145	2230
Xen+Standalone DomU	3996	230	4630
Xen+Standalone DomU stress	4347	465	5210
Real-time software running on R5			
Standalone R5	2568	130	2890
OpenAMP+Standalone	3300	198	3630
OpenAMP+Standalone stress	3521	691	5690

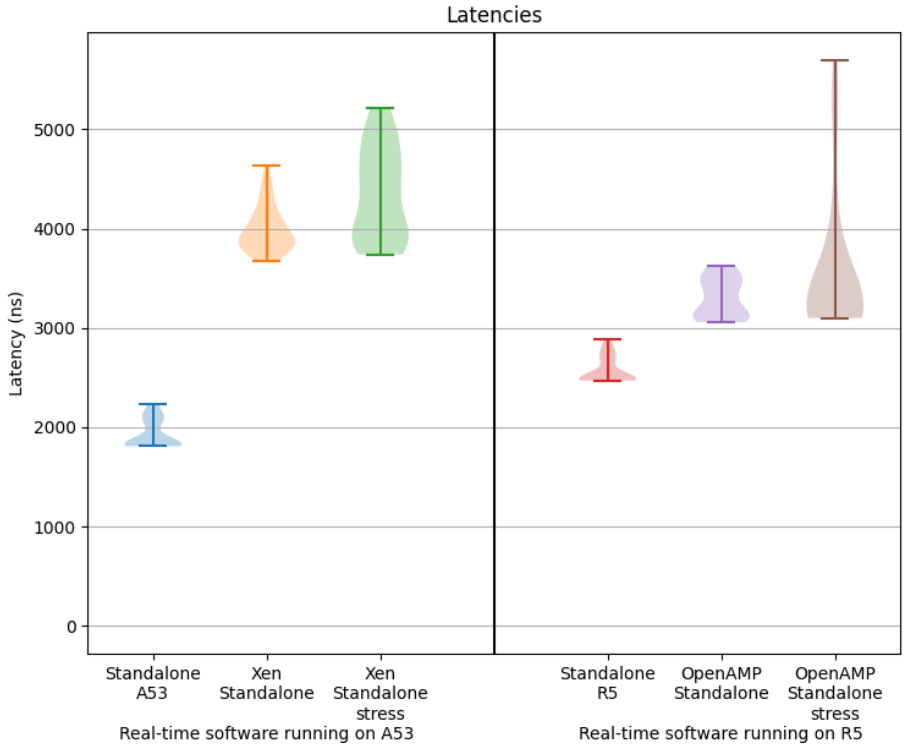

Figure 10. Interrupt latency for 3 to 6 scenarios (Figures 5–8).

First, these results show that the latency is more considerable in the R5 cores (Figure 7) than in the A53 cores (Figure 5) and both are low, in the order of 2 or 3 microseconds. The R5 cores are supposed to be deterministic always, as they are designed to process real-time applications. The maximum or worst-case latencies are not much bigger than the average, so they do not seem problematic. They are low latencies that allow deterministic operation in the range of 3 ms.

Comparing the OpenAMP+Standalone (Figure 8) and the standalone R5 (Figure 7) scenarios shows that the OpenAMP framework increases the latency, although it continues to be low. The maximum value is just 330 ns higher than the average. The Xen hypervisor (Figure 6) also increases the interrupt latency compared to the Standalone A53 scenario (Figure 5). There is a more significant difference (634 ns) between the average and the maximum value, as it is run in the general-purpose cores. The stress affects the deviation, incrementing it in the scenarios with Xen (Figure 6) and OpenAMP (Figure 8), especially the one with OpenAMP.

In scenarios 1 and 2 (Figures 1 and 2), the latencies are not within reasonable limits, but in scenarios 3 to 6 (Figures 5–8) they are. This allows for using these solutions in the endpoints of the smart grid.

8. Conclusions

Smart grid endpoints need a Linux-type OS in the A53 cores for management tasks and a standalone execution or a real-time OS in the R5 cores to make a deterministic use of data from other nodes in the smart grid. The Xen hypervisor and the OpenAMP framework support this, but they may introduce a delay in the system. GOOSE and SV messages used in the smart grid are the most critical and must be generated, transmitted and processed in less than 3 ms. Therefore, it is critical to characterize the delays introduced by Xen and OpenAMP software. There are many studies about the use of hypervisors in embedded systems and the latencies introduced by OSs, however, there are not enough research works about the delays due to hypervisors or multicore frameworks.

In this paper, latencies introduced by the Xen hypervisor are evaluated in Linux systems. The thread latency is measured with the Cyclictest tool. This was carried out without stress and during a test induced by the stress-ng utility. This was carried out in a Linux OS scenario without a hypervisor and in Xen DomU running a Linux OS. The results show that the Linux thread latency is more significant with a hypervisor than without it. IN addition, the hypervisors increase the average delay and the maximum, reaching very high peaks that could be problematic for some applications. The stress test increases these latencies even more. The worst-case latencies obtained, with a statistical average between all the CPU or vCPU maximum latencies, are 47 µs in Linux without stress, 156 µs in Linux with stress, 159 µs in Xen DomU without stress and 104,947 µs in Xen DomU with stress. Using the Xen hypervisor in a stressful situation multiplies the maximum thread latency by 1000. This can be problematic for some applications with real-time requirements, limiting the processes where the latency must be controlled. In real-time systems, the worst-case latency must be kept below specific maximum values, so this increase is very significant. It has been shown that when the Xen hypervisor is used, Linux is not suitable for the smart grid as it does not meet the 3 ms timing constraint.

Then, standalone execution as the real-time part was evaluated, measuring the delay to handle an interrupt created in PL. The measurement was carried out using the integrated logic analyzer (ILA) on the PL side. This measurement was made in six scenarios: (1) a standalone application running on an A53 core, (2) a standalone application running on an R5 one, (3) a standalone application running on a Xen domain while a Linux OS is running on Xen Dom0 without stress, (4) a standalone application running on an R5 while a Linux OS is running on the A53 cores using OpenAMP without stress, (5) a standalone application running on a Xen domain, while a Linux OS is running on stressed Xen Dom0 and (6) a standalone application running on an R5 core, while a Linux OS is running on the A53 cores, using OpenAMP and stressing the A53 cores. In all these cases, the maximum

latency is below 3 ms. This allows the use of those solutions when the 3 ms constraint needs to be ensured in the entire smart grid.

In the future, it would be interesting to evaluate hybrid systems where a Linux OS runs on the A53 cores and FreeRTOS runs on the R5 cores. This way, the interferences that the coexistence of two different OSs in the same device may induce could be analyzed.

Author Contributions: The overall research has been performed by S.A.; Supervision, U.B. and L.M.; Writing—original draft, S.A.; Writing—review and editing, J.L., L.M., U.B. and J.J. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by the Ministerio de Economía y Competitividad of Spain within the project TEC2017-84011-R and FEDER funds as well as by the Department of Education of the Basque Government within the fund for research groups of the Basque university system IT978-16. It has also been supported by the Basque Government within the project HAZITEK ZE-2020/00022 as well as the Ministerio de Ciencia e Innovación of Spain through the Centro para el Desarrollo Tecnológico Industrial (CDTI) within the project IDI-20201264; in both cases, they have been financed through the Fondo Europeo de Desarrollo Regional 2014-2020 (FEDER funds). It has also been supported by the University of the Basque Country within the scholarship for training of research staff with code PIF20/135.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Walls, C. *Multicore Systems: Heterogenous Architectures-Untangling the Technology and Terminology*; Embedded Software; Siemens: Munich, Germany, 2014.
2. Blenk, A.; Basta, A.; Reisslein, M.; Kellerer, W. Survey on Network Virtualization Hypervisors for Software Defined Networking. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 655–685. [CrossRef]
3. Joe, H.; Kang, D.; Shin, J.A.; Dupre, V.; Kim, S.Y.; Kim, T.; Lim, C. Remote graphical processing for dual display of RTOS and GPOS on an embedded hypervisor. In Proceedings of the 2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA), Luxembourg, 8–11 September 2015; pp. 1–4. [CrossRef]
4. Azkarate-askasua, M.; Martinez, I.; Iturbe, X.; Obermaisser, R. Suitability of Hypervisor and MPSoC Architectures for the Execution Environment of an Integrated Embedded System. In Proceedings of the 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, Newport Beach, CA, USA, 28–31 March 2011. [CrossRef]
5. Ko, W.; Yoo, J.; Kang, I.; Jun, J.; Lim, S.S. Lightweight, Predictable Hypervisor for ARM-Based Embedded Systems. In Proceedings of the 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Daegu, Korea, 17–19 August 2016; p. 109. [CrossRef]
6. Jiang, Z.; Audsley, N.C.; Dong, P. BlueVisor: A Scalable Real-Time Hardware Hypervisor for Many-Core Embedded Systems. In Proceedings of the 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Porto, Portugal, 11–13 April 2018; pp. 75–84. [CrossRef]
7. Poggi, T.; Onaindia, P.; Azkarate-askatsua, M.; Grüttner, K.; Fakih, M.; Peiró, S.; Balbastre, P. A Hypervisor Architecture for Low-Power Real-Time Embedded Systems. In Proceedings of the 2018 21st Euromicro Conference on Digital System Design (DSD), Prague, Czech Republic, 29–31 August 2018; pp. 252–259. [CrossRef]
8. Kinebuchi, Y.; Sugaya, M.; Oikawa, S.; Nakajima, T. Task Grain Scheduling for Hypervisor-Based Embedded System. In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC), Dalian, China, 25–27 September 2008; pp. 25–27. [CrossRef]
9. Moratelli, C.; Zampiva, S.; Hessel, F. Full-virtualization on MIPS-based MPSOCs embedded platforms with real-time support. In Proceedings of the 2014 27th Symposium on Integrated Circuits and Systems Design (SBCCI), Aracaju, Brazil, 1–5 September 2014; pp. 1–7. [CrossRef]
10. Xia, T.; Rihani, M.A.F.; Prévotet, J.C.; Nouvel, F. Demo: Ker-ONE: Embedded virtualization approach with dynamic reconfigurable accelerators management. In Proceedings of the 2016 Conference on Design and Architectures for Signal and Image Processing (DASIP), Rennes, France, 12–14 October 2016; pp. 225–226. [CrossRef]
11. Huang, C.H.; Hsiung, P.A. Hardware Resource Virtualization for Dynamically Partially Reconfigurable Systems. *IEEE Embed. Syst. Lett.* **2009**, *1*, 19–23. [CrossRef]
12. Nakajima, T.; Kinebuchi, Y.; Courbot, A.; Shimada, H.; Lin, T.H.; Mitake, H. Composition Kernel: A Multi-core Processor Virtualization Layer for Highly Functional Embedded Systems. In Proceedings of the 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, Tokyo, Japan, 13–15 December 2010; pp. 223–224. [CrossRef]
13. Perez-Botero, D.; Szefer, J.; Lee, R.B. Characterizing hypervisor vulnerabilities in cloud computing servers. In Proceedings of the Workshop on Security in Cloud Computing (SCC), Hangzhou, China, 8 May 2013; pp. 3–10. [CrossRef]

14. Alonso, S.; Lázaro, J.; Jiménez, J.; Muguira, L.; Largacha, A. Analysing the interference of Xen hypervisor in the network speed. In Proceedings of the 2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS), Segovia, Spain, 18–20 November 2020; pp. 1–6. [CrossRef]
15. Rodríguez, M.; Lázaro, J.; Jaime Jiménez, U.B.; Astarloa, A. A Fixed-Latency Architecture to Secure GOOSE and Sampled Value Messages in Substation Systems. *IEEE Access* **2021**, *9*. [CrossRef]
16. Abeni, L.; Faggioli, D. An Experimental Analysis of the Xen and KVM Latencies. In Proceedings of the 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), Valencia, Spain, 7–9 May 2019; pp. 18–26. [CrossRef]
17. Hughes, A.; Awad, A. Quantifying Performance Determinism inVirtualized Mixed-Criticality Systems. In Proceedings of the 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), Valencia, Spain, 7–9 May 2019; pp. 181–184. [CrossRef]
18. Sun, Y.; Li, E.; Yang, G.; Liang, Z.; Guo, R. Design of a Dual-core Processor Based Controller with RTOS-GPOS Dual Operating System. In Proceedings of the 2019 IEEE International Conference on Mechatronics and Automation (ICMA), Tianjin, China, 4–7 August 2019; pp. 1859–1864. [CrossRef]
19. UNE. *Communication Networks and Systems for Power Utility Automation Part 6:Configuration Description Language for Communication in Power Utility AutomationSystems Related to IEDs*; Document IEC 61850-6; UNE: Armidale, Australia, 2018.
20. Hancock, J. How to Choose between a Hypervisor and a Multicore Framework. Tech Design Forums. Available online: https://www.techdesignforums.com/practice/technique/how-to-choose-between-a-hypervisor-and-a-multicore-framework/ (accessed on 4 June 2021).
21. Abeni, L.; Goel, A.; Krasic, C.; Snow, J.; Walpole, J. A measurement-based analysis of the real-time performance of linux. In Proceedings of the Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, San Jose, CA, USA, 27 September 2002; pp. 133–142. [CrossRef]
22. Tafa, I.; Beqiri, E.; Paci, H.; Kajo, E.; Xhuvani, A. The Evaluation of Transfer Time, CPU Consumption and Memory Utilization in XEN-PV, XEN-HVM, OpenVZ, KVM-FV and KVM-PV Hypervisors Using FTP and HTTP Approaches. In Proceedings of the 2011 Third International Conference on Intelligent Networking and Collaborative Systems, Fukuoka, Japan, 30 November–2 December 2011; pp. 502–507. [CrossRef]
23. Lee, J.G.; Hur, K.W.; Ko, Y.W. Minimizing Scheduling Delay for Multimedia in Xen Hypervisor. In *Advanced Communication and Networking*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 96–108. [CrossRef]
24. Pinto, S.; Araujo, H.; Oliveira, D.; Martins, J.; Tavares, A. Virtualization on TrustZone-Enabled Microcontrollers? Voilà! In Proceedings of the 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Montreal, QC, Canada, 16–18 April 2019; pp. 293–304. [CrossRef]
25. Klingensmith, N.; Banerjee, S. Using virtualized task isolation to improve responsiveness in mobile and IoT software. In Proceedings of the International Conference on Internet of Things Design and Implementation, Montreal, QC, USA, 15–18 April 2019; pp. 160–171. [CrossRef]
26. Ahmad, S.; Boppana, V.; Ganusov, I.; Kathail, V.; Rajagopalan, V.; Wittig, R. A 16-nm Multiprocessing System-on-Chip Field-Programmable Gate Array Platform. *IEEE Micro* **2016**, *36*, 48–62. [CrossRef]
27. Xilinx. Libmetal and OpenAMP User Guide (UG1186). 2019. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug1186-zynq-openamp-gsg.pdf (accessed on 4 June 2021).
28. Powell, A.A. Performance of the Xilinx Zynq System-on-Chip Interconnect with Asymmetric Multiprocessing. Ph.D. Thesis, Temple University Graduate Board, Philadelphia, PA, USA, 2014. [CrossRef]
29. Govindan, S.; Choi, J.; Nath, A.R.; Das, A.; Urgaonkar, B.; Sivasubramaniam, A. Xen and Co.: Communication-Aware CPU Management in Consolidated Xen-Based Hosting Platforms. *IEEE Trans. Comput.* **2009**, *58*, 1111–1125. [CrossRef]
30. Toumassian, S.; Werner, R.; Sikora, A. Performance measurements for hypervisors on embedded ARM processors. In Proceedings of the 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, India, 21–24 September 2016; pp. 851–855. [CrossRef]
31. Zeng, L.; Wang, Y.; Feng, D.; Kent, K.B. XCollOpts: A Novel Improvement of Network Virtualization in Xen for I/O-Latency Sensitive Applications on Multicores. *IEEE Trans. Netw. Serv. Manag.* **2015**, *12*, 163–175. [CrossRef]
32. Xilinx. PetaLinux Tools. Developer Tools. Embedded Software. Available online: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html (accessed on 4 June 2021).
33. Redhat. Finding Realtime Linux Kernel Latencies. Tools for Measuring Latency. Available online: http://people.redhat.com/williams/latency-howto/rt-latency-howto.txt (accessed on 4 June 2021).
34. Abeni, L.; Faggio, D. Using Xen and KVM as real-time hypervisors. *J. Syst. Archit.* **2020**, *106*, 101709, [CrossRef]