technologies

Article

Specific Electronic Platform to Test the Influence of Hypervisors on the Performance of Embedded Systems

Jaime Jiménez [1], Leire Muguira [1], Unai Bidarte [1], Alejandro Largacha [2] and Jesús Lázaro [1,*]

1 Electronics Technology Department, Faculty of Engineering of Bilbao, University of the Basque Country, 48013 Bilbao, Spain; jaime.jimenez@ehu.eus (J.J.); leire.muguira@ehu.eus (L.M.); unai.bidarte@ehu.eus (U.B.)
2 GE Renewable Energy, 48170 Zamudio, Spain; alejandro.largachaferreras@ge.com
* Correspondence: jesus.lazaro@ehu.eus

Abstract: Some complex digital circuits must host various operating systems in a single electronic platform to make real-time and not-real-time tasks compatible or assign different priorities to current applications. For this purpose, some hardware–software techniques—called virtualization—must be integrated to run the operating systems independently, as isolated in different processors: virtual machines. These are monitored and managed by a software tool named hypervisor, which is in charge of allowing each operating system to take control of the hardware resources. Therefore, the hypervisor determines the effectiveness of the system when reacting to events. To measure, estimate or compare the performance of different ways to configure the virtualization, our research team has designed and implemented a specific testbench: an electronic system, based on a complex System on Chip with a processing system and programmable logic, to configure the hardware–software partition and show merit figures, to evaluate the performance of the different options, a field that has received insufficient attention so far. In this way, the fabric of the Field Programmable Gate Array (FPGA) can be exploited for measurements and instrumentation. The platform has been validated with two hypervisors, Xen and Jailhouse, in a multiprocessor System-on-Chip, by executing real-time operating systems and application programs in different contexts.

Keywords: virtualization; hypervisor; virtual machines; multi processor; multi-operating-system

check for updates

Citation: Jiménez, J.; Muguira L.; Bidarte U.; Largacha A.; Lázaro, J. Specific Electronic Platform to Test the Influence of Hypervisors on the Performance of Embedded Systems. *Technologies* **2022**, *10*, 65. https://doi.org/10.3390/technologies10030065

Academic Editor: Spyridon Nikolaidis

Received: 25 April 2022
Accepted: 23 May 2022
Published: 24 May 2022

1. Introduction

Many complex digital circuits must host different operating systems in a single electronic platform to synchronize real-time and deferrable tasks or give different priorities to running applications. Applications around Industry 4.0 or the Industrial Internet of Things require incorporating more non-real-time functionality and common software architectures such as RTOperating System (OS) and bare-metal-based periodic control loops. Hence, some hardware–software codesign techniques—named virtualization—must be exploited to run all the operating systems autonomously of each other, as confined in different processors or virtual machines.

There are different levels of virtualization or virtualization schemes, full virtualization, para-virtualization, and static partitioning or core virtualization [1]. Due to the strict constraints on resources and the computing power of embedded environments, not all are suitable for industrial embedded systems. Considering the prioritization of real-time operations and the restriction of computing resources in embedded systems, static partitioning is more suitable for them. In this case, because of the straightway coupling between the physical resources and the virtualized environments, the real-time operation determinism is less affected. So, partitioning is better suited to embedded resources than full virtualization [2].

These virtualization schemes require an underlying software—named the Virtual Machine Monitor (VMM) or the so-called hypervisor—to handle the guest machines or guest OS. The hypervisor runs at a privileged execution level, and with the help of virtual systems, it manages the sharing of underlying physical resources. However, at the same time, the hypervisor introduces an additional layer between the outside world and the processing unit, i.e., a potentially longer latency. The common use cases for hypervisors in embedded systems are the consolidation, legacy operating systems and multiple security levels [3].

A type 1 hypervisor runs directly on the hardware. It hosts OS and manages resources and memory allocation for the virtual machines. In a type 2 hypervisor or guest OS virtualization, the hypervisor runs over a host OS, the lowest layer of software, which provides drivers and services for the hypervisor hosting virtualized guest OS. Comparing both types of hypervisors, type 1 introduces less overhead, providing a better performance and greater capacity to optimize the allocation of hardware resources to the various virtual machines. Access to the devices does not depend on the driver being virtualized compared to the type 2 hypervisors.

Consequently, as an interface, the hypervisor may determine the effectiveness of the system when reacting to events, depending not only on how quickly it switches the working context but also on which hardware resources it occupies or releases. Nevertheless, measuring, estimating, or comparing the performance when configuring the virtualization in different ways is not trivial. Specific electronic testbenches to host various processors, install distinct operating systems, configure the hardware–software partition and show significant merit figures are demanded.

To characterize and compare multiple virtualization interferences, the authors have designed a specific hardware-testbench. The programmed application measures the additional latency introduced in an interrupt service during normal execution, and it is regarded as fairly general and of capital importance in industrial applications. In many such systems, for example, smart grid applications, the time to react in front of a stimulus is the key factor of the equipment [4]. In such a system, a baremetal application provides the greatest speed with the smallest flexibility. Nevertheless, a compromise point may be the division of work by the use of a hypervisor. It provides, at the same time:

- Flexibility of a higher level operating system;
- Careful control of latencies of a baremetal application;
- Security by separation of hardware and software between both elements.

Following the structure of this article, Section 2 presents a brief survey of related work about other testbench setups. Section 3 specifies the considered electronic platform for testing different hypervisors. The proposed approach for an automated test procedure is discussed in Section 4, and the paper ends with some conclusions.

2. Hardware-Testbenches for Hypervisors

So far, the performance of hypervisors has been mostly tested in computer platforms; sometimes of the personal type [5–15], others in servers [16–26]; multi-purpose embedded systems have also been used in the same way [1,27,28]. Although we could have exploited these complex testbenches, and all the advantages of high-level software interfaces, sensors, and electronic instrumentation in general, would have had to be attached outside. Besides, reconfiguring the setup implied manual changes, and external connections and instruments introduced errors in the latencies' measurements. Reference [29] used an FPGA platform in a similar context; not to test how the hypervisor slowed down the performance of the processor system, like us, but to analyze the interference between virtual machines in their working-isolation. Table 1 shows in which electronic cards the referenced teams performed the tests.

Table 1. Other research works on measuring the performance of hypervisors in electronic boards.

Platform	References
Computer	[5–15]
Servers	[16–26]
Embedded	[1,27,28]
FPGA	[29]

Although the authors of [29] conducted some research on CPU performance in an FPGA card with a chip similar to ours—Xilinx ZCU102—they were not interested in the interference of the hypervisor on latencies but in using Jailhouse as the tool to isolate different virtual machines—so that they do not disturb each other's resources.

Bansal et al. [30] also refer to performance in a Xilinx ZCU102, but, in this case, the authors propose Jailhouse at the end as the solution to the problems found about memory contentions: isolating every subsystem using the hypervisor. Hence, to the best of our knowledge, this present work is the first to propose a hardware configurable testbench for analyzing the latency introduced by a hypervisor, exploiting, at the same time, the reconfigurable part of the FPGA to create the instrumentation.

3. The Electronic Platform

The main contribution of this work is designing and implementing a hardware platform valid for testing flexibly and automatically characterizing different hypervisors in various working configurations. Our objective is to offer a versatile electronic tool oriented to evaluate the influence on the system's response due to the use of hypervisors. These are the main criteria in order to select the hardware platform:

- Two of the most used hypervisors, Xen and Jailhouse, have been selected as the reference for the design requirements; they maintain a list of hardware systems on which they have ever been tested. The Xen list is longer because it is an older alternative. For the sake of generality, the selected hardware should appear in both lists. If it does not, the adequacy of it should be considered feasible;
- To test the virtualized systems, tools to generate the Linux cell are demanded, with its kernel, device-tree and file-system, and the baremetal cell. The selected solution must provide this software, and the availability of the source code is also valuable, in case modifications were needed.

An Advanced RISC Machines (ARM)-based architecture has been chosen because it is the most used one for embedded systems, and it is present in the Processing System (PS) of Zynq and Zynq UltraScale+ families in Xilinx devices. It is also suitable for the task since ARMv8 architecture provides virtualization extensions in hardware.

The Xilinx Zynq UltraScale+ family has been chosen because it includes ARMv8 architecture in PS and a Programmable Logic (PL) part where application-specific circuits can be implemented in order to create a test setup. The Xilinx ZCU102 [31] board, which includes a Zynq UltraScale+ Multiprocessor Programmable System-on-a-Chip (MPSoC) chip, appears in Xen and Jailhouse lists. It would have been a good option, but the research team has been using the UltraZed [32] board for the last works, and it was concluded that, as it includes a chip of the same family, slight modifications would be needed in order to execute both hypervisors so that this electronic platform has been chosen. When preparing this work, the UltraZed card was not among Jailhouse's list of supported and tested platforms; there was no cell file for it, so the files corresponding to the root cell and the cell with the baremetal application were created. There is a configuration file for every cell (in our case, one for the root cell and one for the baremetal one). These files created for the UltraZed contain the configuration structure that Jailhouse needs to locate the different resources of the platform on the memory map. Making possible the use of the Ultrazed board for Jailhouse is another valuable contribution of this work.

The complete hardware system comprises two boards: the processing UltraZed-EG System-On-Module (SOM) and the UltraZed Input Output (IO) carrier card. These are the main circuits:

- UltraScale+ MPSoC XZU3EG-1SFVA625E, which includes:
  - 4 ARM Cortex-A53 (ARMv8) cores (up to 1.2 GHz)
  - 2 ARM Cortex-R5 cores (up to 500 MHz)
  - Mali-400 MP2 graphic processor (up to 600 MHz)
  - 154K Logic Cells
  - 141K flip-flops
  - 7.6 Mb RAM
  - 360 DSP blocks
- 2 GB DDR4 SDRAM
- 64 MB QSPI flash
- 8 GB eMMC flash
- 1 Gigabit Ethernet
- 1 SD card
- 12 Peripheral Module interface (PMOD) in PL
- 2 USB-UARTs
- 1 JTAG
- 8 switches and 8 LED in PL

Hardware System Description for the UltraScale+ MPSoC

The four ARM Cortex-A53 (ARMv8) cores support the processing of both guests, the Linux system, and the baremetal. The PL part is used as general IO and to measure the timing of the system. Figure 1 shows the block diagram, as it is designed in Vivado.

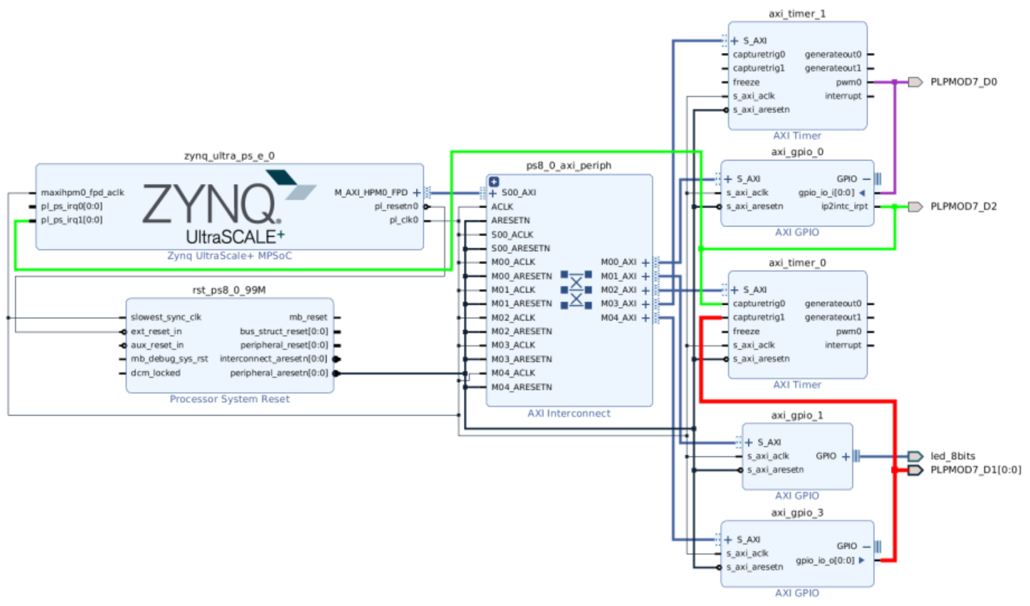

Figure 1. Block diagram designed in Vivado. The AXI interconnect is shared among all processors, which means that it can be accessed both by the hypervisor and the standalone application. A timer is in charge of generating periodic interrupts while another interrupt measures the latency.

It is a straightforward system in which all the blocks are connected by the Advanced eXtensible Interface (AXI) bus and appear mapped in microprocessor memory. The objective is to measure the latency that hypervisors introduce in treating interrupts in guest baremetal systems. For this purpose, an instrumentation subsystem has been created; to do that, the following blocks and connections between them have been arranged:

- axi_gpio_0 [33]: generates an interrupt whenever a change occurs in its input. The output is connected as an interrupt generated in PL and used as input in PS;
- axi_timer_1 [34]: it is configured as Pulse-width modulation (PWM), and its output is the input signal in axi_gpio_0;
- axi_gpio_3: its output signal is activated from the interrupt service routine;
- axi_timer_0: it captures two events: the interrupt event generated by axi_gpio_0 and the activation of axi_gpio_3, which occurs when the interrupt is attended. So the difference between both times is the time needed to attend the interrupt;
- axi_gpio_1: its output is connected to a LED in order to visually follow the execution.

The interrupt events are also monitored in an external logic analyzer using a PMOD connector as the backup instrumentation.

4. Test Procedure

The main purpose of the hardware described in the previous section is to provide a flexible and configurable means to accurately and transparently measure the latency in executing an interrupt service function. In order to be explored by different test cases, the following steps have been automated:

- Create a base application-level operating system. In our case, Linux. For that purpose, we require:
  - Create a kernel with hypervisor supports;
  - Add the hypervisor executable and configuration;
- Create a base low latency application. In our case, a baremetal application has been created for the interrupt service function. In a more general way, a real-time application could be used, both baremetal or RTOS;
- Create a periodic interrupt by the use of the hardware timer;
- Capture the time to serve the interrupt by the use of the hardware capture module;
- Repeat the operation for a statistically significant number of times to obtain the data under different CPU loads. The hypervisor's operating system is stressed to test the impact of the hypervisor in the high-priority baremetal application.

It must be stated that the test must be repeated to filter out outlying values. Simultaneously, these outliers are of great importance since, in real-time systems, the worst-case scenario is the defining scenario. An overall structure can be seen in Figure 2.

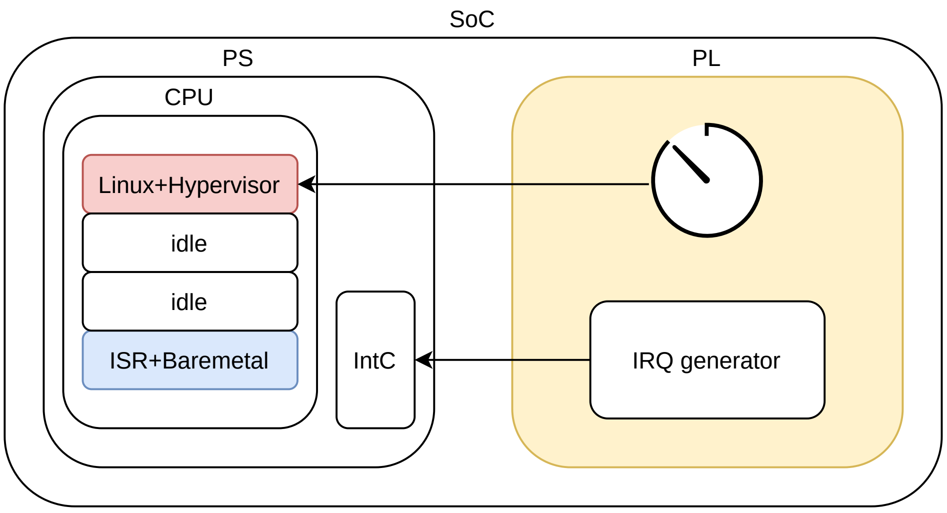

Figure 2. An overall structure of the proposed scenario. There is a hardware periodic interrupt generator, which is paired with a timer. The interrupt sets the timer while the ISR stops it.

4.1. General Flow

To test different hypervisors, a procedure has been devised (Figure 3). The hardware can generate continuous interrupts. What is left is to measure the time to service those interrupts under different scenarios. To do that, an Interrupt Service Routine (ISR) has been created as a baremetal application. This ISR toggles the stop signal of the capture counter through a GPIO. A program inside the Linux cell/Dom reads the start and stop times, saving them into a file. This ISR resides in a baremetal application so that it does not depend on any operating system. To generate different scenarios, the stress_ng tool is used inside the Linux cell/Dom. The output values are transmitted to a computer and analyzed.

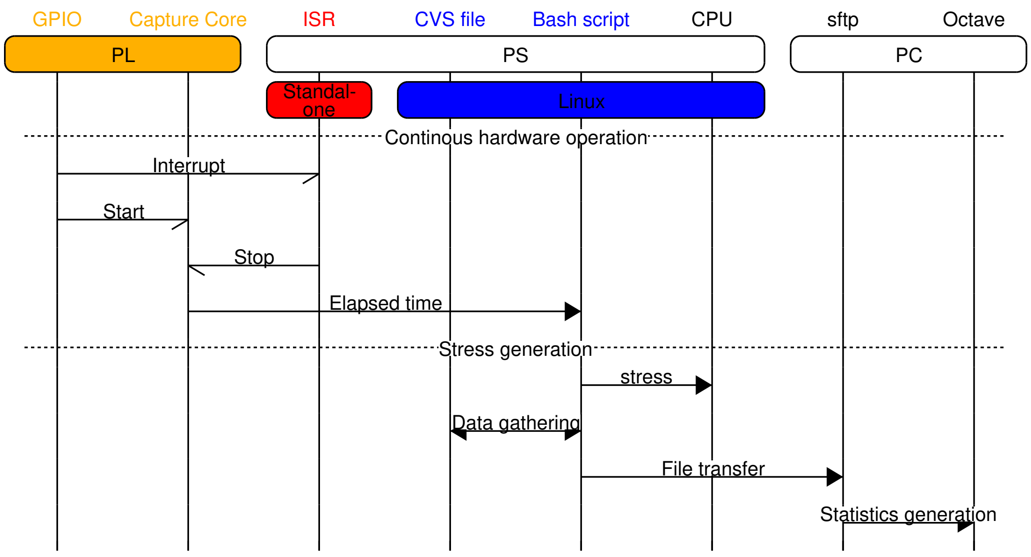

Figure 3. General flow. In the PL, the timer is started while the interrupts are generated. The ISR receives the interrupt and stops the timer. The processor also reads the timer value and generates all the files for the statistical evaluation. At the same time, the processor also stresses the memory and CPU. In an external PC, all the data are gathered and processed.

The test procedure has been automated to a great degree. A script generates the different stress levels (in our case, CPU stress, memory stress, and CPU and memory stress), reads the ISR latency, and generates a result file for every stress test. Each file contains several hundred instances of the interrupt (the number can be configured to get a statistically significant number of instances); this file is transmitted using sftp to a host PC where the data are analyzed and graphs generated using an Octave/Gnuplot script.

All these scripts are highly configurable, and stress_ng has many different options. Depending on the final application, different stress patterns can be generated. We have selected two common baseline scenarios, although personalized ones could be as easily created.

This automation is useful since it can be exploited as an in-system self-test. Instead of using artificial stress software, the actual final application can be used while in the field to measure possible problems.

4.2. Validation Test

In order to validate our setup, we have tested two popular hypervisors: Xen and Jailhouse. At the same time, as mentioned, we have stressed the operating system managing the hypervisor. The stress has been focused on CPU and memory usage.

For example, the platform automatically generated 700 rounds of the interrupt in three different scenarios and two different hypervisors—a sample can be seen in Figures 4 and 5.

The first experiment (see Figure 4) has been performed, stressing the CPU; the second (see Figure 5), the virtual memory.

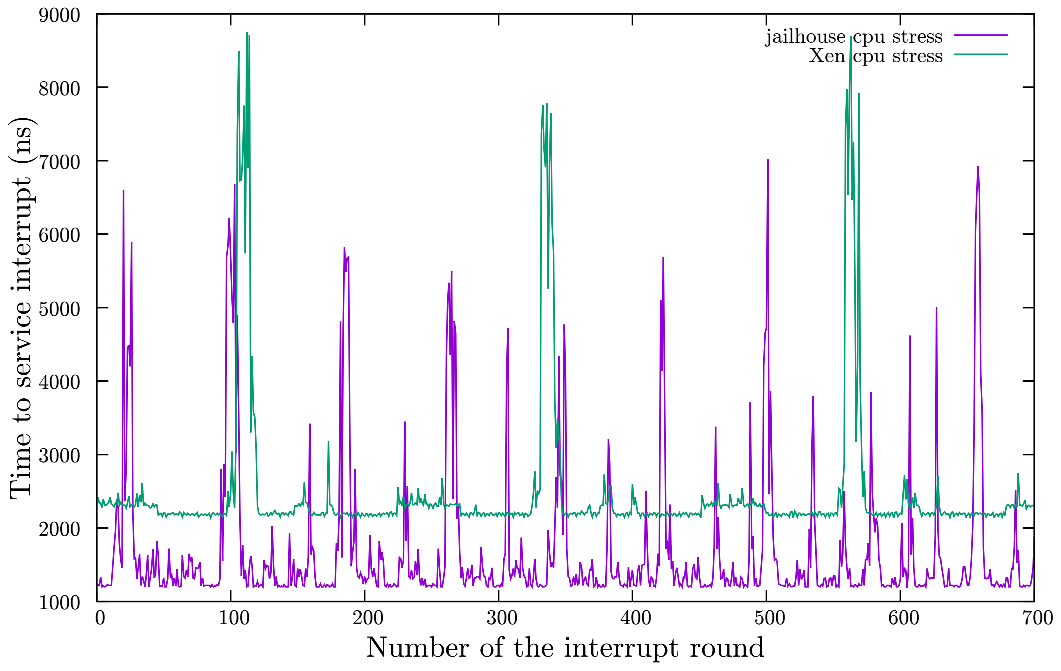

Figure 4. Results for Xen and Jailhouse when CPU is stressed. Overall, Jailhouse has a better baseline, although many more spikes than Xen. These spikes correspond to transients in the hypervisor.

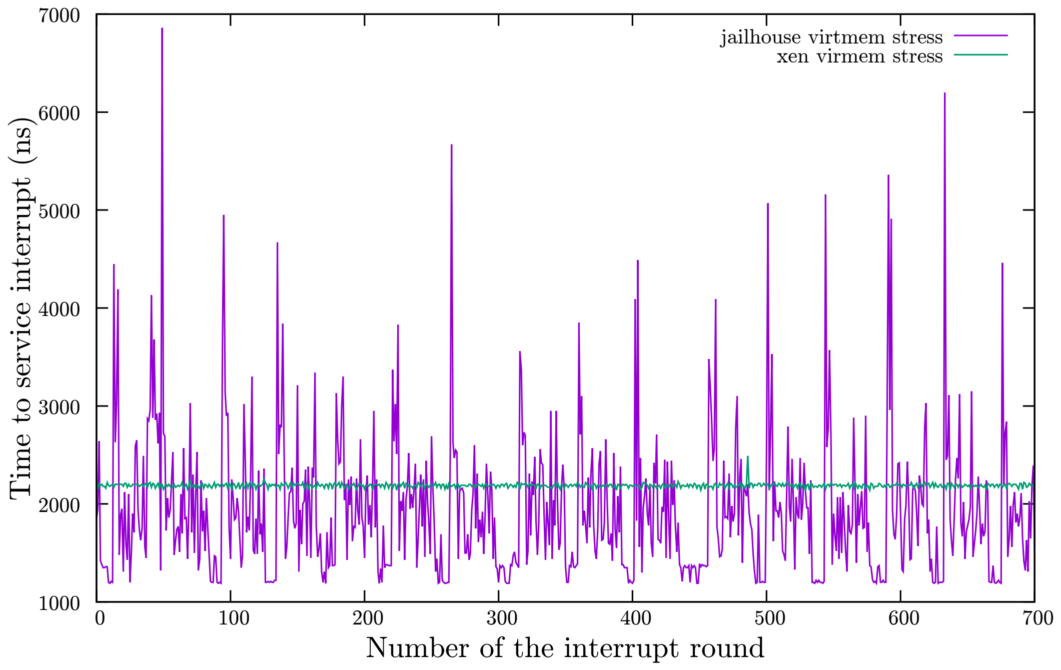

Figure 5. Results for Xen and Jailhouse when the memory module is stressed.

The tool also sources a summary of statistical data—for example, see Table 2. In the optimal case, a latency of approximately 1 µs for Jailhouse and 2 µs for Xen were found.

Table 2. Statistical results in the time to service interrupt (ns) for all 700 interrupt rounds.

	CPU Stress		**Virtmem Stress**		**All Stress**	
Statistics	**Jailhouse**	**Xen**	**Jailhouse**	**Xen**	**Jailhouse**	**Xen**
min	1190.0000	2140.0000	1190.0000	2140.0000	1190.0000	2140.0000
Q1	1220.0000	2190.0000	1420.0000	2180.0000	1200.0000	2190.0000
median	1320.0000	2210.0000	1790.0000	2190.0000	1310.0000	2220.0000
Q3	1580.0000	2310.0000	2230.0000	2200.0000	1460.0000	2310.0000
max	7020.0000	8750.0000	6860.0000	2490.0000	7050.0000	8750.0000
mean	1705.7489	2481.8545	1934.0942	2188.4593	1611.0699	2337.4608
StdDev	1033.8568	1022.1210	721.2251	21.4788	943.2967	556.9779
skewness	3.0235	4.5639	2.2143	3.7068	3.5246	7.8376
kurtosis	11.8676	23.2601	11.0569	57.1830	15.4045	71.8205

5. Conclusions

In this work, a flexible, configurable, and automated platform composed of an electronic system based on a complex FPGA has been proposed and validated for measuring and evaluating the performance of different hypervisors.

First, we investigated the impact of the hypervisor's performance. The integration of virtualization techniques allows the coexistence of real-time and non-real-time applications running in different operating systems inside a single digital circuit. In this virtualized context, the hypervisor is responsible for monitoring and managing the independent operating systems, deciding which takes control of hardware resources in each moment. Therefore, the effectiveness of the system in the presence of interrupts or events is connected to the hypervisor behavior straightaway. We have noticed that the selected platforms for previous studies in most cases are computer ones, but they did not take advantage of reconfigurable computing solutions technologies. Besides, they do not analyze the I/O contention, such as general-purpose I/O interrupts.

A hardware testbench composed of an SoC that hosts some processors, installs various operating systems, configures the hardware–software partition, and shows significant merit figures has been designed and implemented. A test procedure has been defined and automated to a great degree to measure the latency in the execution of an interrupt service function, and the setup has been validated using two type 1 hypervisors: Xen and Jailhouse.

Author Contributions: Conceptualization, A.L.; methodology, J.J., J.L. and A.L.; validation, U.B.; investigation, L.M. and A.L.; data curation, J.L. and A.L.; writing—original draft preparation, A.L.; writing—review and editing, J.J., U.B. and J.L.; supervision, J.L.; project administration, J.J. and J.L.; funding acquisition, J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work has 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 and FEDER funds.

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

References

1. 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; IEEE: Piscataway, NJ, USA, 2016. [CrossRef]
2. Kou, E. Virtualization for Embedded Industrial Systems, 2018. SPRY317A. Available online: https://www.ti.com/lit/wp/spry317b/spry317b.pdf (accessed on 25 April 2022).
3. Greenbaum, J.; Garlati, C. Hypervisors in Embedded Systems. Applications and Architechtures. In Proceedings of the Embedded World Conference, Nuremberg, Germany, 27 February–1 March 2018.
4. *IEC 61850-1 ed2.0*; Communication Networks and Systems for Power Utility Automation—Part 1: Introduction and Overview. IEC: Geneva, Switzerland, 2013.
5. Polenov, M.; Guzik, V.; Lukyanov, V. Hypervisors Comparison and Their Performance Testing. In *Advances in Intelligent Systems and Computing*; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 148–157._16. [CrossRef]

6. Poojara, S.R.; Ghule, V.B.; Birje, M.N.; Dharwadkar, N.V. Performance Analysis of Linux Container and Hypervisor for Application Deployment on Clouds. In Proceedings of the 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), Belgaum, India, 21–22 December 2018; IEEE: Piscataway, NJ, USA, 2018. [CrossRef]
7. Ferreira, R.D.F.; de Oliveira, R.S. Cloud IEC 61850: DDS Performance in Virtualized Environment with OpenDDS. In Proceedings of the 2017 IEEE International Conference on Computer and Information Technology (CIT), Helsinki, Finland, 21–23 August 2017; IEEE: Piscataway, NJ, USA, 2017. [CrossRef]
8. Graniszewski, W.; Arciszewski, A. Performance analysis of selected hypervisors (Virtual Machine Monitors—VMMs). *Int. J. Electron. Telecommun.* **2016**, *62*, 231–236. [CrossRef]
9. Pesic, D.; Djordjevic, B.; Timcenko, V. Competition of virtualized ext4, xfs and btrfs filesystems under type-2 hypervisor. In Proceedings of the 2016 24th Telecommunications Forum (TELFOR), Belgrade, Serbia, 22–23 November 2016; IEEE: Piscataway, NJ, USA, 2016. [CrossRef]
10. Vojtesek, J.; Pipis, M. Virtualization of Operating System Using Type-2 Hypervisor. In *Advances in Intelligent Systems and Computing*; Springer International Publishing: Berlin/Heidelberg, Germany, 2016; pp. 239–247._22. [CrossRef]
11. Bujor, A.C.; Dobre, R. KVM IO profiling. In Proceedings of the 2013 RoEduNet International Conference 12th Edition: Networking in Education and Research, Iasi, Romania, 26–28 September 2013; IEEE: Piscataway, NJ, USA, 2013. [CrossRef]
12. Yang, Z.; Fang, H.; Wu, Y.; Li, C.; Zhao, B.; Huang, H.H. Understanding the effects of hypervisor I/O scheduling for virtual machine performance interference. In Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, Taipei, Taiwan, 3–6 December 2012; IEEE: Piscataway, NJ, USA, 2012. [CrossRef]
13. Binu, A.; Kumar, G.S. Virtualization Techniques: A Methodical Review of XEN and KVM. In *Advances in Computing and Communications*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 399–410._40. [CrossRef]
14. Zhang, L.; Bai, Y.; Luo, C. idsocket: API for Inter-domain Communications Base on Xen. In *Algorithms and Architectures for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 324–336._29. [CrossRef]
15. Xu, X.; Zhou, F.; Wan, J.; Jiang, Y. Quantifying Performance Properties of Virtual Machine. In Proceedings of the 2008 International Symposium on Information Science and Engineering, Shanghai, China, 20–22 December 2008; IEEE: Piscataway, NJ, USA, 2008. [CrossRef]
16. Dordevic, B.; Timcenko, V.; Kraljevic, N.; Davidovic, N. File system performance comparison in full hardware virtualization with ESXi and Xen hypervisors. In Proceedings of the 2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, Bosnia and Herzegovina, 20–22 March 2019; IEEE: Piscataway, NJ, USA, 2019. [CrossRef]
17. Yang, Z.; Liu, C.; Zhou, Y.; Liu, X.; Cao, G. SPDK Vhost-NVMe: Accelerating I/Os in Virtual Machines on NVMe SSDs via User Space Vhost Target. In Proceedings of the 2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2), Paris, France, 18–21 November 2018; IEEE: Piscataway, NJ, USA, 2018. [CrossRef]
18. Sok, S.W.; Jung, Y.W.; Lee, C.H. Optimizing System Call Latency of ARM Virtual Machines. *J. Phys. Conf. Ser.* **2017**, *787*, 012032. [CrossRef]
19. Zhang, D.; Wu, H.; Xue, F.; Chen, L.; Huang, H. High Performance and Scalable Virtual Machine Storage I/O Stack for Multicore Systems. In Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, China, 15–17 December 2017; IEEE: Piscataway, NJ, USA, 2017. [CrossRef]
20. Ul Ain, Q.; Anwar, U.; Mehmood, M.A.; Waheed, A. HTTM—Design and Implementation of a Type-2 Hypervisor for MIPS64 Based Systems. *J. Phys. Conf. Ser.* **2017**, *787*, 012006. [CrossRef]
21. Twardowski, M.; Pastucha, E.; Kolecki, J. Performance of the Automatic Bundle Adjustment in the Virtualized Environment. In Proceedings of the 2016 Baltic Geodetic Congress (BGC Geomatics), Gdansk, Poland, 2–4 June 2016; IEEE: Piscataway, NJ, USA, 2016. [CrossRef]
22. Borislav, Đ.; Nemanja Maček, V.T. Performance Issues in Cloud Computing: KVM Hypervisor's Cache Modes Evaluation. *Acta Polytech. Hung.* **2015**, *12*, 147–165. [CrossRef]
23. Ye, K.; Wu, Z.; Zhou, B.B.; Jiang, X.; Wang, C.; Zomaya, A.Y. Virt-B: Towards Performance Benchmarking of Virtual Machine Systems. *IEEE Internet Comput.* **2014**, *18*, 64–72. [CrossRef]
24. Tang, X.; Zhang, Z.; Wang, M.; Wang, Y.; Feng, Q.; Han, J. Performance Evaluation of Light-Weighted Virtualization for PaaS in Clouds. In *Algorithms and Architectures for Parallel Processing*; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 415–428._32. [CrossRef]
25. Cacheiro, J.L.; Fernández, C.; Freire, E.; Díaz, S.; Simón, A. Providing Grid Services Based on Virtualization and Cloud Technologies. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 444–453._50. [CrossRef]
26. André Bögelsack, H.K.; Wittges, H. Performance Overhead of Paravirtualization on an Exemplary Erp System. In Proceedings of the 12th International Conference on Enterprise Information Systems, Funchal, Portugal, 8–12 June 2010; SciTePress—Science and and Technology Publications: Setubal, Portugal, 2010. [CrossRef]
27. Kloda, T.; Solieri, M.; Mancuso, R.; Capodieci, N.; Valente, P.; Bertogna, M. Deterministic Memory Hierarchy and Virtualization for Modern Multi-Core Embedded Systems. In Proceedings of the 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Montreal, QC, Canada, 16–18 April 2019; IEEE: Piscataway, NJ, USA, 2019. [CrossRef]
28. Heiser, G.; Leslie, B. The OKL4 microvisor. In Proceedings of the First ACM Asia-Pacific Workshop on Workshop on Systems—APSys'10, New Delhi, India, 30 August 2010; ACM Press: New York, NY, USA, 2010. [CrossRef]

29. Danielsson, J.; Seceleanu, T.; Jagemar, M.; Behnam, M.; Sjodin, M. Testing Performance-Isolation in Multi-core Systems. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019; IEEE: Piscataway, NJ, USA, 2019. b7 [CrossRef]
30. Bansal, A.; Tabish, R.; Gracioli, G.; Mancuso, R.; Pellizzoni, R.; Caccamo, M. Evaluating memory subsystem of configurable heterogeneous MPSoC. In Proceedings of the 14th Annual Workshop onOperating Systems Platforms for Embedded Real-Time Applications, Barcelona, Spain, 3 July 2018.
31. Xilinx. *Zynq UltraScale+ MPSoC ZCU102*; Technical Report; Xilinx: San Jose, CA, USA, 2020.
32. Xilinx. *Zynq UltraScale+ MPSoC Data Sheet: Overview*; Technical Report; Xilinx: San Jose, CA, USA, 2019.
33. Xilinx. *AXI GPIO v2.0 LogiCORE IP Product Guide*; Technical Report; Xilinx: San Jose, CA, USA, 2016.
34. Xilinx. *AXI Timer v2.0 LogiCORE IP Product Guide*; Technical Report; Xilinx: San Jose, CA, USA, 2016.